Dialectical Nonmonotonic Inheritance

Arthur M. Farley

CIS-TR-96-14
December 1996

Department of Computer and Information Science
University of Oregon

Dialectical Nonmonotonic Inheritance

Arthur M. Farley

Computer and Information Science
University of Oregon
Eugene, OR 97403 USA

ABSTRACT

We present a new model of nonmonotonic inheritance based on dialectical argumentation over mixed inheritance networks. A mixed inheritance network consists of strict and default links between nodes representing classes of objects. An object is represented by strict or default connections to nodes of an inheritance network. Rules for constructing allowable arguments are presented. A defeat relation between argument pairs is defined in terms of argument strengths. We represent the vulnerability of elements of the defeat relation to impacts from other arguments. Notions of defendable and justifiable arguments are developed. A flexible decision procedure for determining acceptability of an inheritance claim is defined in terms of burden of proof. We apply our model to examples from the inheritance reasoning literature, demonstrating its key properties. An appendix demonstrates our Scheme implementation on further examples.

Key Words: inheritance reasoning, argumentation, burden of proof

Introduction

Inheritance reasoning systems have been studied in artificial intelligence as means for representing and reasoning about object-related classification and property knowledge. A desire for correct and efficient representation and inference has motivated the study of several effective approaches to determining inheritability of property and class knowledge. Early work on semantic networks (Quillian,1968; Sowa, 1984, 1992) introduced a structural, procedural basis for this type of reasoning. Under this approach, class and property concepts are represented as nodes in a network interconnected by subclass and instance-of (i.e. IS-A) links. Properties or parts of an object or class of objects that are shared with (almost) all elements of a superclass need not be repeated, but instead are inherited as the result of a search process through ancestors in the inheritance network. NETL has been one of the most famous instantiations of this network-based approach to inheritance, including a parallel implementation (Fahlman, 1979; Fahlman, et.al., 1981). Subsequent, direct approaches to inheritance reasoning have maintained the structure of semantic networks as a basic, underlying, theoretical element, endeavoring to define inheritability semantics formally in terms of graph-theoretic properties of substructures in the networks. Uncertainty has been introduced into the representation by allowing non-strict (or default) links.

The procedural semantics associated with semantic networks was actively criticized by Woods (1975) among others. This criticism resulted in a number of attempts to formalize the semantics of inheritance networks in a more denotational fashion. One general approach has been to translate inheritance networks into equivalent sets of logical statements, thereby leaving behind the structural/procedural elements of semantic networks and grounding their meaning in terms of more syntactic, logic-based semantics. Explorations with this logical approach has resulted in the definition and application of various nonmonotonic and default logics to capture the meaning of non-strict relations among taxonomic categories (Etherington, 1988; Reiter, 1980; Ginsberg, 1987).

In this paper, we introduce a new perspective on inheritability reasoning, that of dialectical argumentation. Under this approach, arguments for and against a claim are considered before a decision is taken. We employ the direct approach to the representation of inheritance knowledge and associated argument structures, grounding our semantics in terms of interactions between arguments represented as paths found in an inheritance network. We describe a dialectical framework that is consistent with desired features of inheritance semantics. Evaluating the appropriateness of inheritability results produced by other approaches has proven to be a difficult, intuitive matter. One advantage that an argumentation perspective has is that conflicting indications, which pose a problem for logic-based reasoning systems, are just the sort of situations that dialectical argumentation is meant to address and resolve. A set of examples that illustrates many of the underlying issues has been presented recently (Horty, 1994); a selection of these will be used to demonstrate and evaluate our dialectical approach later in the paper.

In the next section, we formalize our definition of inheritance networks. Then, we turn our attention to defining the notion of argument as it applies in the setting of direct, inheritance

reasoning. We characterize the defeat relation between arguments of a network. We then consider the question of, given an inheritance network and an associated object description, what inheritance claims are supported with respect to that object. We propose that this decision be relative to a flexible burden of proof, rather that fixed as a property of a sceptical or credible reasoning system. Finally, we demonstrate our model and our Scheme implementation on a number of examples that highlight elements of our theory.

Inheritance Networks

We take the knowledge underlying inheritance reasoning to be an inheritance network, as specified in a form derived from that presented by Horty (Horty, 1994). Nodes in an inheritance network, denoted by (words of) small letters, are used to represent classes of objects. We use a small letter from the end of the alphabet (e.g., x, y, z) to represent an arbitrary node of a given inheritance network. Classes of objects correspond to sets of objects having a common designation (e.g., dog) or sharing common properties or parts (e.g., are brown or have four legs).

Nodes of an inheritance network are interconnected by directed links, each link connecting two nodes of the network. There are four types of links, corresponding to possible combinations of strength {strict, default} and sign {positive, negative} labels. The link types ==> and =/=> denote strict positive and strict negative links, respectively. These are equivalent in meaning to universally quantified, conditional statements in first-order predicate logic. The link x==>y represents that every individual object of kind x is of kind y (i.e., "All x's are y's"). The link x=/=>y represents no object of kind x is of kind y (i.e., "All x's are not y's". This link implies its inverse; we sometimes use the doubly-directed arrow, i.e., <=/=>, to indicate this symmetry.

Link types --> and -/-> denote default positive and default negative links, respectively. The meaning of a default link is not expressible in terms of standard predicate logic. It captures various notions of imperfect generalization that are often made in commonsense reasoning. Informally, the link x-->y (x-/->y) represents the notion expressed by such sentences as: "Most x's are (not) y's" or "By default or usually, x's can be considered (not) to be y's". In general, default links represent a strong, but not universal, inheritance relationship between the two classes of objects. As an example, bird --> fly could capture the notion that "birds usually fly". Default links are understood to admit the presence of relatively few, known or expected, exceptions. The intuitive semantics associated with the above informal statements can vary. This ambiguity in meaning must be resolved in a formal manner. Under our approach, the formal meaning of such links will be defined in terms of their allowed combination with other links in forming arguments, their impact on resultant argument structures, and their effects on the outcome of arguments.

Default links have been called "defeasible" links (Pollock, 1987), as they can be defeated, or preempted, by more specific or conclusive information. We will use the term "default" for such links to capture the strong connection that is intended. We reserve the term "defeasible" for a more general, somewhat weaker relation that exists between nodes interconnected by argument paths involving several default links in an inheritance network, i.e., the default relation is not transitive in

our approach. We will discuss the construction of such defeasible arguments in the next section.

Objects will be denoted by special nodes labeled by (words of) capital letters; we will use a capital letter from the end of the alphabet (e.g., X, Y, Z) to represent an arbitrary object node. Object nodes have no incoming links from the inheritance network. An object node is connected to an inheritance network by the same types of links as discussed above, representing what we know directly about the given individual object. When connecting an object X to a node y by a strict link, i.e., X==>y (X=/=>y), we are representing that X definitely is (is not) of kind y. If we connect X to y by a default link, i.e., X-->y (X-/->y), we are representing that X most likely or usually is (is not) of kind y; this leaves open the possibility that the object X may not have the represented relationship to the kind y. Again, the meaning of this default link will be defined formally in terms of how it combines to form inheritance arguments.

We will use the knowledge represented by an inheritance network as basis for answering questions regarding acceptable classifications of given objects. A particular class of interest, i.e., a *goal*, is simply a node in the inheritance network. We want to decide whether to accept a proposed inheritance relation between a given object and the goal class. This acceptance will be determined by constructing arguments for and against a proposal and considering their interactions. As noted above, what is known directly about an object is represented by a set of strict and default links connecting its corresponding node to nodes of an inheritance network. Such direct arguments are extended by construction of indirect arguments based on the identification of appropriate, longer paths in the inheritance network

We provide an example of an inheritance network with an object connected in Figure 1. This network could be interpreted as representing three interclass relationships: that most birds are flying things, that all penguins are birds, and that all penguins do not fly. We also represent an object TWEETY, which we are sure is a bird and is most likely a penguin. We use heavy lines to represent strict links and thin lines to represent default links in all figures to follow.

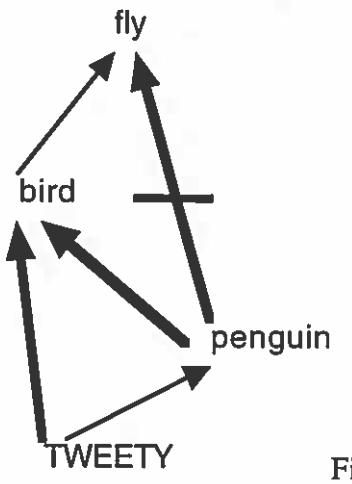

Figure 1

Arguments

We can consider arguments from two perspectives. First, an argument can be seen as a structured knowledge entity, as when we say "give me a good argument" for some claim. From this perspective, each inheritance argument provides a basis for believing that a given object does or does not belong to a given goal class. In this paper, we develop a model of argument structure as paths and their interactions in inheritance networks. How arguments are generated and compared is the second perspective on an argument, that of argument as dialectical process, as when we say "we had quite an argument" over some claim. In this paper, we develop a dialectical approach to argumentation whereby arguments for and against a proposed inheritability relationship are generated alternately by opposing sides, each side responding to the other's arguments, until one side must concede and a decision is reached. Deciding which side must respond and whether a side has succeeded in turning the argument back to the other side or if it must concede will depend upon a burden of proof, another key element of our model of argumentation.

We first define the structure of allowable arguments derived from an inheritance network, discussing their properties and their interactions, then define an associated inheritability semantics based upon several burdens of proof, and finally outline a process of dialectical argumentation that is consistent with the semantics and sensitive to burden of proof.

Argument Structure

Given an inheritance network and an associated object description, we are interested in defining the notion of arguments for and against membership of the given object in a particular class of the network. An *inheritance argument* is a directed path in an inheritance network, possible starting at an object node. An inheritance argument P from *start* node x to *finish* node y through an intermediate, possibly empty, sequence of intermediate nodes π, denoted as $P(x, \pi, y)$, is such that if node v immediately precedes node u in the argument then v is directly connected to u by a link of the inheritance network. Not all such paths in an inheritance network constitute allowable inheritance arguments, however. Which paths form allowable arguments will be defined by argument construction rules below.

We characterize an argument in terms of its strength {strict, default, defeasible} and its sign {positive, negative}. The informal meanings of strict and default arguments are the same as were discussed earlier for links in inheritance networks. We will use *defeasible* strength for arguments between nodes interconnected by compound argument paths of particular forms; defeasible arguments capture an inheritance relationship that is weaker than that established by strict or default links. If $P(x, \pi, y)$ is a defeasible positive argument, it represents the notion that "there is reason or it is reasonable to believe that x's are y's". If this path begins at an object node X, we are saying that "X could reasonably be considered to be a y". It represents the notion that there is some indication, by the given argument, that the inheritance relationship holds. The support is

weaker than a default argument provides, however. The formal semantics of defeasible arguments will be determined in terms of their interactions with arguments of other strengths. We will see that distinguishing between default and defeasible argument strengths will allow us to capture, in a straightforward manner, certain inheritability semantics that have been judged previously to be correct. We will demonstrate these points through examples later in the paper.

Allowable arguments in an inheritance network, with their associated strengths and signs, are defined recursively in the "backward" direction from a given finish node. Links in the network form direct arguments, as defined by the following argument construction rule:

Rule R1: (direct arguments)

A. given x ==>y, P(x , ø, y) is a strict positive argument
B. given x =/=>y, P(x , ø, y) is a strict negative argument
C. given x -->y, P(x , ø, y) is a default positive argument
D. given x -/->y, P(x , ø, y) is a default negative argument
E. given x<==y, P(x, ø, y) is a candidate negative argument

In the above rule, x refers to an arbitrary class x or object X; the symbol "ø" represents the empty sequence of nodes. Subrule R1(E) captures the potential for use of modus tollens reasoning (i.e., ~x==>~y) over a strict (logically sufficient) positive link in an inheritance network. If a negative link is added as "head" link of a candidate negative argument, the argument will become a negative argument of appropriate strength. A candidate negative argument that is not completed by such a forward-directed, negative link does not, in and of itself, constitute an allowable argument.

We extend an argument path by adding a new start node and head link to a given argument, creating a compound, or indirect, argument, as defined by the following construction rule:

Rule R2: (indirect arguments)

A. P(x,π,y) is a strict positive path not containing z, then
   (i) given z==>x, P'(z,x,π,y) is a strict positive argument
   (ii) given z-->x, P'(z,x,π,y) is a default positive argument
B. P(x,π,y) is a strict negative argument not containing z, then
   (i) given z==>x, P'(z,x,π,y) is a strict negative argument
   (ii) given z-->x, P'(z,x,π,y) is a default negative argument
C. P(x,π,y) is a candidate negative argument not containing z (z), then
   (i) given z<==x, P'(z,x,π,y) is a candidate negative argument
   (ii) given z =/=> x, P'(z,x,π,y) is a strict negative argument
   (iii) given z -/-> x, P'(z,x,π,y) is a default negative argument
D. P(x,π,y) is a default positive argument not containing Z (z), then
   (i) given z==>x, P'(z,x,π,y) is a defeasible positive argument
   (ii) given Z==>x, P'(Z,x,π,y) is a default positive argument.
   (iii) given z-->x, P'(z,x,π,y) is a defeasible positive argument

E. P(x,π,y) is a default negative argument not containing Z (z), then
(i) given z==>x, P'(z,x,π,y) is a defeasible negative argument
(ii) given Z==>x, P'(Z,x,π,y) is a default negative argument
(iii) given z-->y, P'(z,x,π,y) is a defeasible negative argument

F. P(x,π,y) is a defeasible positive argument not containing z, then
(i) given z==>x, P'(z,x,π,y) is a defeasible positive argument
(ii) given z-->x, P'(z,x,π,y) is a defeasible positive argument

G. P(x,π,y) is a defeasible negative argument not containing z, then
(i) given z==>x, P'(z,x,π,y) is a defeasible negative argument
(ii) given z-->x, P'(z,x,π,y) is a defeasible negative argument

Throughout rule R2, z represents either an arbitrary class z or object Z.

Subrules R2(A)(i) and R2(B)(i) capture the standard, forward chaining rules of implication in first-order logic. Subrules R2(A)(ii) and R2(B)(ii) indicate that a default link followed by a strict argument is a default argument. Subrule R2(C) represents ways in which a candidate negative argument can be either transformed into a strict or default negative argument or simply extended in its present status. Note that a negative argument either ends with a single negative link or contains a single negative link followed by a candidate negative argument. The two final subrules, R2(F) and R2(G), indicate allowable extensions of defeasible arguments.

Subrules R2(D) and R2(E) represent that either a strict or a default extension of a default argument between classes becomes a defeasible argument. This captures the observation that even when most x's are (or are not) y's and all or most z's are x's, the z's that are x's may not tend to be the ones that are (or are not) y's. However, it is reasonable to think that that they may be; thus, we assign a defeasible strength of the resultant argument. In fact, the actual relationship between z and y may be anything from strict to empty. Members of the class z can not be assumed to be uniformly spread across the class x; as such, the strength of the argument between z and y is weakened to defeasible, reflecting the increased uncertainty or lack of knowledge.

These two subrules also indicate that when an object node Z is connected to a class node of a network by a strict link, we can assume a uniform distribution of occurrence over the class and, thus, can adopt the strength of the existing argument for the result. By these two subrules, we distinguish the meaning of "is-instance-of" for a link from an object to a class of an inheritance network from the meaning of "is-subclass-of" for a link between classes within an inheritance network. These subrules form a critical element in our theory of inheritance argument structure, distinguishing our approach from others that fail to note a difference between default and defeasible strength arguments and between links into and within an inheritance network. These distinctions allow us to handle certain argument comparisons in a more straightforward, intuitive manner, as will be demonstrated later.

Given an inheritance network I, an object X, and a goal node y, an argument constructed by the rules above is *grounded* if it starts with X, finishes with y, and is based on the links within

I and a link from X into I. For example, given the inheritance network in Figure 1, we find three grounded arguments, as follows:

against :: P_1(TWEETY, penguin, fly);

for :: P_2(TWEETY, penguin, bird, fly), P_3(TWEETY, bird, fly).

We see there are arguments both for and against the claim that TWEETY is a member of the class of objects that fly. Arguments can stand in a conflicting, or contradictory, relationship to each other. Conflict relationships between arguments have been defined in a number of previous works on argumentation (Sartor, 1993; Pollock, 1987; Loui, 1987; Sartor, 1993; Dung, 1995). We specialize these more general definitions to address our current context of inheritance reasoning based upon a direct representation of inheritance knowledge. We will discuss the relationship of our work to recent work in argumentation later in the paper.

Two inheritance arguments P(x, π1, y) and P'(x, π2, y), having the same start and finish nodes, *directly conflict* if they differ in sign. More generally, two arguments *conflict* if one argument directly conflicts with a subargument of the other, i.e., one argument is of the form (π_1, x, π_2, y, π_3) and the other is the form (x, π, y), where (x, π, y) and (x, π_2, y) are of opposite sign. In our example, we find two pairs of (directly) conflicting grounded arguments: (P_1, P_2) and (P_1, P_3); the ungrounded argument P_4(penguin, $\emptyset$, fly) also conflicts with P_2 and P_3.

We expect conflicting arguments in many real-world circumstances. However, some conflicts reflect inconsistencies in knowledge. Two arguments within an inheritance network (i.e., not involving an object node) are *inconsistent* if they directly conflict and are both either strict or default in strength. Assuming that only non-empty classes of significant size occur in inheritance networks, if we find an argument concluding that all or most x's are y's and another concluding that all or most x's are not y's, there is clearly an inconsistency in our knowledge. With a grounded argument starting at an object, conflicting default arguments are not necessarily inconsistent. The object, as a single element, could occur in subparts of conflicting default classes so as not to be inconsistent. However, two strict, grounded arguments that are in conflict indicate an inconsistency in object-related knowledge, given a consistent inheritance network.

To determine the outcome of an argumentation process, arguments for and against a claim must be generated and compared. A key notion in argumentation theory is that conflicts between arguments can lead to defeat of arguments. In the context of inheritance networks, a conflict between two arguments of differing strengths results in the defeat of the argument of lesser strength. One argument A *defeats* (is a *defeater* of) an argument B iff argument B is of the form (π_1, x, π_2, y, π_3), A is of the form (x, π, y), argument A and the subargument (x, π_2, y) of B are of opposite sign, and argument A is stronger than subargument (x, π_2, y).

The transitive relation "stronger than" has the obvious definition for our inheritance arguments: strict arguments are stronger than default arguments, which, in turn, are stronger than defeasible arguments. In our example above, argument P_1 defeats argument P_2. An argument A that conflicts with but does not defeat an argument B (i.e., is of the same strength) *rebuts* (is a

rebuttal of) argument B. In our example, argument P_1 rebuts argument P_3

Our definition of argument defeat subsumes several, previous definitions of preemption in mixed inheritance networks, as recently reviewed (Horty, 1994). The definition of mixed preemption proposed there depends on the following definition of strict inheritance relations. For a node z and inheritance network I, define $\mu(z)$ to be node z plus those nodes x in I for which there exists a positive strict argument from z to x; similarly, define $\underline{\mu(z)}$ to be those nodes x in I for which a strict negative argument from z to x exists. These sets represent, respectively, the strict positive and strict negative classifications that z inherits from the network. *Mixed preemption* is defined (as derived from Horty, 1994), as follows:

A positive path P(x, π, u, y), where u-->y, is *preempted* if there exists nodes v and m, such that (i) either v = x or P is of the form P(x, π1, v, π2, u, y) and (ii) either (a) v-/->m and m is an element of μ(y) or (b) v-->m and y is an element of $\underline{\mu(m)}$. A negative path P(x, π, u, y), where u-/->y, is *preempted* if there exists nodes v and m, such that (i) either v = x or P is of the form P(x, π1, v, π2, u, y) and (ii) v-->m and y is an element of μ(m).

It is easy to see that our definition of defeat between arguments subsumes this definition. In preemption of a positive path P, (ii)(a) depends on extension of a candidate negative path by a negative head link from v to m, while (ii)(b) extends an existing strict, negative path with a positive head from v to m. In either case, the argument from v through m to y is of default strength, while the argument from v to y through u is at most of defeasible strength. Thus, the positive path is defeated under our model. A similar analysis holds for the case of defeat of a negative path. Our definition of defeat is more general than the definition of mixed preemption given above, applying as well when the link between v and m is strict. Furthermore, it captures situations when the link between u and y and the link between v and m are both strict, as long as the path from v to u is of default strength or less. In these latter cases, the arguments though m are of strict strength, while those though u are at most of default strength. While conflicts between arguments of such strengths would be considered inconsistent if between classes, as discussed above, our definition of defeat can still resolve the conflict when allowed from a object, i.e., when v = x is an object.

Our definition of defeat is not consistent with the notions of argument subsumption and associated notions of off-path and on-path preemption (Sandewall, 1986; Touretzky, et.al, 1987). These approaches to preemption view longer, more complex arguments having the same start and finish nodes as further, in-depth specifications of the arguments; as such, they consider these arguments to be more convincing or stronger and allow preemption to occur by these paths. According to our notion of defeat, based upon comparisons of argument strengths, longer arguments that involve default links lead to arguments that are most often of lesser, defeasible strength. As such, they are vulnerable to defeat by more direct arguments of default or strict strength. The subsumption-based approaches to preemption have seen limited success and application. It requires a link semantics fundamentally different from the normal meaning that reflects an underlying set-theoretic or probabilistic outlook based on class membership and overlap.

Our definition of defeat, based upon the comparison of conflicting argument strengths and

recognition of subarguments, has simplified and generalized earlier preemption definitions. However, our present definition, as well as those previously proposed, fails to represent a certain vulnerability in the defeat relation. This will be demonstrated by the following example, adapted from Horty (1994), regarding Hermann, as shown in Figure 2.

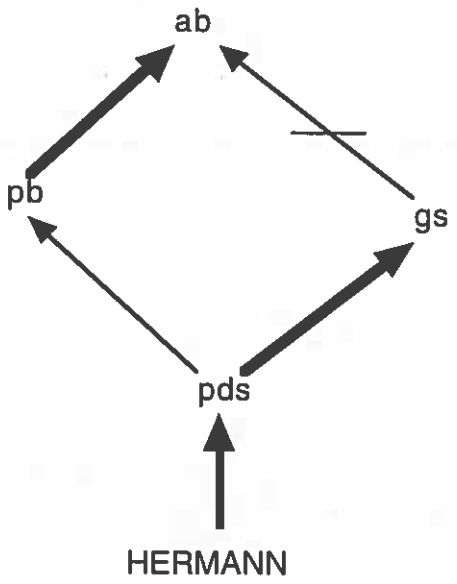

Figure 2

Here, we know that Hermann is a Pennsylvania Dutch speaker (pds), that all Pennsylvania Dutch speakers are German speakers (gs) and most are born in Pennsylvania (pb), that those born in Pennsylvania are born in America (ab), and, finally, that most German speakers are not American born. The defeasible strength argument P(HERMANN, pds, gs, ab) for the claim that Hermann is not born in America is defeated by the default strength argument P'(pds, pb, ab).

Suppose we learn that Hermann is not born in Pennsylvania, as represented in Figure 3. In this case, while P' does (directly) defeat the subargument P''(pds, gs, ab) of P, it is no longer reasonable to consider that P itself is defeated. We would be basing defeat of that argument on an argument that requires support for a node (in this case pb) that we are saying is not true for Hermann. To address this issue, we extend our definition of the defeat relation between two arguments to include an associated set of *vulnerable arguments*, being those that must not be defeated if the given defeat relation is to be effective.

If argument $P_1(\pi_1, x, \pi_2, v, \pi_3)$, where π_1 is a sequence of one or more nodes, is defeated by an argument $P_2(x, \pi, y)$, where π is a sequence of one or more nodes, in inheritance network I, then the set of arguments starting at nodes of π_1 and finishing at nodes of π passing through nodes along paths P_1 and P_2 constitute the set of *vulnerable arguments* for the defeat relation between P_2 and P_1. Argument P_2 defeats P_1 in I only if none of the vulnerable arguments associated with the defeat relation are defeated in I. The vulnerable arguments of a defeat relation

are those that start at nodes occurring prior to the beginning of the defeating argument and end at intermediate nodes of the defeating argument. From this definition, an argument that directly defeats another argument has no vulnerable arguments.

In the example of Figure 3, the positive argument P'''(HERMANN, pds, pb) is a vulnerable argument for the defeat relation between P' and P. The argument P''' is defeated by the strict, direct, negative argument P''''(HERMANN, ø, pb), which thereby counters the defeat of P. While existence of a defeat relation between two conflicting arguments can be determined directly by comparing their strengths, the effectiveness of the defeat will depend upon the overall structure of the inheritance network. This leads us to our discussion of what arguments and what claims actually are supported in an inheritance network for a given object representation.

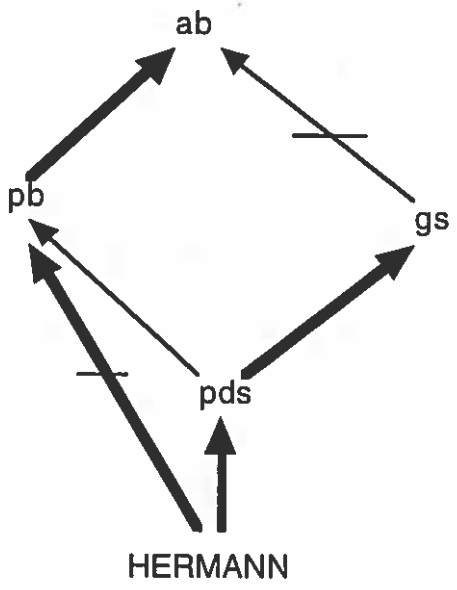

Figure 3.

Burden of Proof and Inheritance Semantics

Given that arguments for and against a set of inheritance claims can be generated and compared, noting defeats and rebuttals among them, the question remains as to how we can determine the ultimate inheritability semantics, i.e., which claims are accepted and which are not for a given inheritance network and object representation. An important element of this decision is a consideration of how conservative we want to be when choosing to accept or reject an inheritance claim. We could consider the relative strengths of arguments for and against the inheritance claim, as suggested by the above definition of defeat. We also could consider the type of error, that of commission (i.e., false positives or acceptances) or omission (i.e., false negatives or rejections), we are more willing to live with should we be wrong. How can we capture these dimensions of concern in a decision procedure that determines argument outcomes?

Research to date on inheritance reasoning has resulted in several proposals for defining so-called credulous and skeptical reasoning systems (Horty, et.al., 1990; Touretzky, et.al., 1987). There has been considerable discussion and disagreement as to the definitions for such systems, due to semantic difficulties that have arisen under the differing proposals. Our goal here is not to discuss these definitions and associated semantic issues, as this has been done elsewhere (Horty, 1994). Rather, we will present an alternative approach, based upon the comparison of arguments for and against an inheritance claim and upon an allocation of risks in a flexible manner through explicit specification of a burden of proof as parameter to the argument process. We will contrast our approach with prior definitions when appropriate and possible below.

An *inheritance claim* is a statement as to the membership of a given object in a given class. A positive inheritance claim between an object X and class y will be denoted as X~~>y, while X~/~>y will denote the complementary, negative claim. Each grounded argument of the form P(X, π, y) *supports* one of these two claims. Whether a claim is acceptable will depend upon the support it has and whether, as a result, it can win a dialectical argument under a given burden of proof. We also will allow inheritance claims between classes, representing class inclusion.

One domain for which the notion of burden of proof has been defined and applied previously is the domain of legal argumentation. Different burdens of proof are mandated at differing stages of a legal process or for differing types of legal action, as a means for allocating risks and costs of error in the legal process. For example, arguments sufficient to indict someone in a criminal proceeding need not be as convincing as those needed ultimately to convict that person of the criminal offense. When considering conviction in criminal cases, we are more concerned with errors of commission (i.e., finding someone guilty when they are not) and, thus, place a relatively high burden of proof on the side arguing for guilt. The burden of proof is made less for criminal indictment, as a trial will follow to correct any error at that phase. In tort cases, where we are arguing over the possible award of damages in civil suits, concern over risk of error is less than in criminal circumstances, and, thus, the burden of proof is reduced, as well. Importantly, the person making the tort claim bears the burden of justifying any award; thus a tie goes to the defendant.

As suggested above, there are two aspects to a specification of burden of proof: (i) which side of a claim (i.e., positive or negative) bears the burden and (ii) what level of proof is required. The first aspect addresses whether we are more concerned about accepting false positives (errors of commission), in which case the burden is placed on the positive claim, or false negatives (errors of omission), where the burden is placed on the negative claim. For example, in some circumstances, we may require a good argument supporting a particular claim before we are willing to accept it. In others, when accepting a claim may have high positive value and little risk of loss is perceived should an error occur, we may demand, instead, a good argument against the claim before denying its acceptance.

The second aspect of burden of proof, that of proof level, addresses the issue of what constitutes a good or convincing argument in a given circumstance. Strength of argument, as we have defined it above for inheritance arguments, is one aspect of the notion of proof level we seek.

However, proof level also must address the existence of conflicts between arguments. As such, our definitions will be based upon the following notions of defendable and justifiable arguments.

A *defendable argument* is an inheritance argument that cannot be defeated according to the given inheritance network and object representation. We can ask that a good argument be more than simply defendable, however. A *justifiable argument* is a defendable argument with the added requirement that every conflicting argument can be defeated (i.e., is not defendable). An argument A that conflicts with an argument B is either a defeater or a rebuttal of B. As such, a justifiable argument is a defendable argument that has no defendable rebuttals.

We can define three proof levels that a claim must meet to win an argument when bearing the burden of proof in terms of our notions of defendable and justifiable arguments, as follows:

- *scintilla of evidence (se)*: there exists a defendable argument supporting the claim;
- *preponderance of evidence (pe)*: there exist more defendable arguments in support of the claim than in support of its negation;
- *dialectical validity (dv)*: there exists a justifiable argument supporting the claim.

Scintilla of evidence is clearly the weakest proof level, requiring only that a defendable argument exist for the given claim; it ignores the existence of defendable rebuttals of arguments supporting the claim. Preponderance of evidence occupies the middle ground, allowing defendable rebuttals only if they are outweighed by arguments in favor of the claim. In the case of inheritance arguments, this only can occur by having more defendable arguments for the claim. At the other extreme, dialectical validity does not abide existence of any defendable rebuttals for at least one argument supporting the claim; in other words, all arguments conflicting with at least one supporting argument for the claim must be defeated. We borrow the names of legal burdens of proof, as this aspect of our model is inspired by legal tradition. Our burdens of proof clearly differ from those applied in the law, but do reflect an increasing stringency of requirements for winning an argument. Legal notions of burden of proof extend to substantive and procedural policies related to actual trials that do not apply in our model of argumentation in inheritance networks.

We define the semantics associated with a given inheritance network and object description in terms of the sets of acceptable inheritance claims under a given burden of proof as defined by the above three proof levels. We define these sets of claims in terms of sets of defendable and justifiable arguments.

We denote by *D(I)* the set of defendable inheritance arguments within a given inheritance network I. We denote by *D(I, X)* the set of grounded inheritance arguments that are defendable, given inheritance network I and object representation X. Similarly, we denote by *J(I)* the set of justifiable inheritance arguments in a given inheritance network I and by *J(I,X)* the set of grounded inheritance arguments that are justifiable, given inheritance network I and object representation X. By our above definitions, D(I) (D(I, X)) is contained in J(I) (J(I, X)).

We denote by *C(I, X, L)* the set of inheritance claims that meet the requirements of proof level L in inheritance network I for object representation X. The set C(I, X, L) is derived from

sets D(I,X) and J(I,X), as follows:

An inheritance claim X~~>y (X~/~>y) is an element of C(I, X, se) iff there exists a positive (negative) argument of the form (X, π, y) in D(I, X).

An inheritance claim X~~>y (X~/~>y) is an element of C(I, X, pe) iff there exists more positive (negative) arguments of the form (X, π, y) than arguments of the same form with opposite sign in D(I, X) .

An inheritance claim X~~>y (X~/~>y) is an element of C(I, X, dv) iff there exists a positive (negative) argument of the form (X, π, y) in J(I, X).

It is clear that the three proof levels defined above result in a hierarchy of acceptable inheritance claims based on set inclusion. For a given inheritance network I and object representation X, C(I, X, se) contains C(I, X, pe), which in turn contains C(I, X, dv).

For our example regarding TWEETY presented in Figure 1, we have the following sets of acceptable claims for the various burdens of proof:

C(I, TWEETY, se) =
{TWEETY~~>penguin, TWEETY~~>bird,
TWEETY~~>fly, TWEETY~/~>fly};
C(I, TWEETY, pe) = C(I, TWEETY, dv) =
{TWEETY~~>penguin, TWEETY~~>bird}.

If P(z, π, y) is an allowable argument in an inheritance network I where z is either an object node or a node of I and y is a node of I, then any subsequence of (z, π, y) is a *subargument* of P(z, π, y). A subargument of argument A is simply a connected subpath of argument A. A *prefix* of an argument P(z, π, y) is a subargument of P(z, π, y) that starts with z.

We have the following *prefix inclusion property* for the sets of defendable and justifiable arguments: If argument P(X, π, y) is an element of D(I, X) (J(I, X)) then all prefixes of P(X, π, y) are also elements of D(I, X) (J(I, X)).

As a result, the corresponding claim sets C(I, X, se) and C(I, X, dv) have an analogous *prefix claim inclusion property*, stated as follows: Given claim c in C(I, X, se) (C(I, X, dv)) with supporting argument P(X, π, y) in D(I, X) (J(I, X)), for every z in π, the corresponding claim X~~>z or X~/~>z is in C(I, X, se) (C(I, X, dv)). The corresponding suffix inclusion property for subarguments ending with y or for corresponding claims of the form z~~>y or z~/~>y is not true in general. This will be shown clearly in the third example of Appendix I, demonstrating the impact of the defeat of vulnerable arguments upon the effectiveness of defeats in an inheritance network.

These prefix inclusion properties are not true for proof level *pe* as defined, however. Under preponderance of evidence, a prefix argument could be outweighed by rebutting arguments for some class on the argument path, while the overall argument still stands. This situation is demonstrated in Figure 4. In the example, there is only one allowable argument for claim A~~>g; the argument is defendable, as no conflicting argument defeats it or any subargument. However, there are two defeasible arguments supporting A~/~> f and only one supporting A~~>f. Thus, A~/~>f is included in C(I, A, pe), while A~~>f is not. As such, A~~>g is a "dangling claim" for

preponderance of evidence, in the sense that, while it is in the set of claims for pe, there exists no argument supporting it that has every subargument in the set of arguments for pe. Appendix I presents results for this inheritance network as computed by our implementation in Scheme.

This "anomaly" is due to the fact that a negative argument for a subargument claim does not extend (forward) as an allowable negative argument in support of another claim. Such propagation over a subsequent link in the inheritance network, even if allowed, would only be a weak, abductive application of the link, i.e., given x~~>y, essentially argue that ¬x~~>¬y. This weak argument, even if allowed, would be defeated by any direct, and thus stronger, application of the link. To overcome this problem, we can define *strong preponderance of the evidence* (*spe*) to be the proof level of preponderance of the evidence with the added constraint that every prefix of every supporting defendable argument also be acceptable under *spe*. This would eliminate the dangling claims that are otherwise allowed under preponderance of evidence as it is defined.

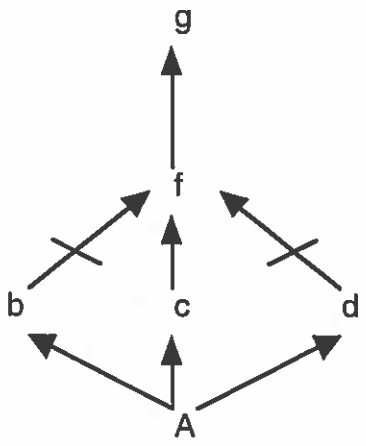

Figure 4.

Examples

We now turn our attention to a number of examples that demonstrate general principles of our approach and illustrate the impacts that burden of proof has upon argument semantics.

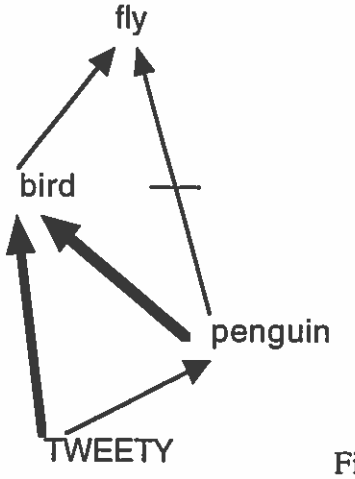

Figure 5

We first consider variations on our TWEETY example of Figure 1. If we weaken the negative connection from penguin to fly to be only a default link, as presented in Figure 5, we have quite a different situation than that originally posed. Now the positive claim has two defendable arguments, including P_3 of default strength; the negative claim can only muster a single argument P_1, now of defeasible strength) that is defeated by P_3. Here, we use argument names established earlier for particular argument paths in the network. Hence, the positive claim TWEETY~~>fly can win arguments up through dialectical validity.

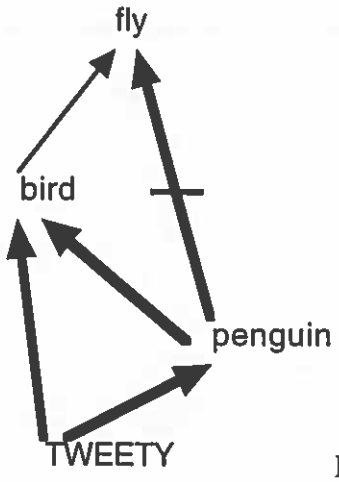

Figure 6.

In Figure 6, we change Figure 1 to have a strict link between TWEETY and penguin, as this example is most often posed in the literature. We see that we can generate a strict argument for the negative claim, but only default and defeasible arguments in support of the positive claim. As such, the negative claim TWEETY~/~>fly can win arguments up through dialectical validity.

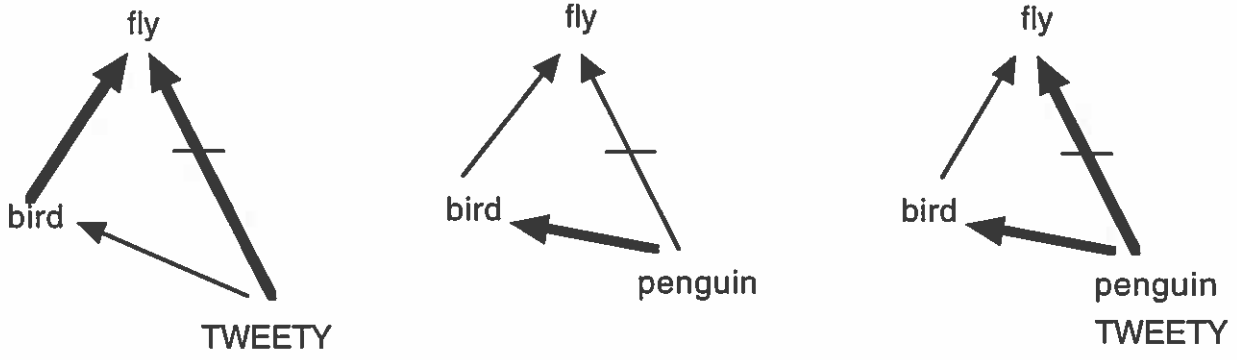

Figure 7.

These examples deal indirectly with issues of specificity in inheritance (Poole, 1985). It has been argued consistently that more specific information should override less specific indications. From our perspective on inheritance reasoning, this means more direct arguments

should defeat less direct arguments, depending, of course, on the link types involved and the strengths of resultant arguments. Figure 7 shows three examples of a "specificity triangle".

In these three examples, the direct negative link from penguin or TWEETY to fly, representing the more specific information, defeats the indirect, positive arguments through the node for birds. In the first case, we are fairly certain that TWEETY is a bird and that all birds fly, yet we know specifically that TWEETY does not fly. Note that this network would be inconsistent if TWEETY were replaced by a class node, such as penguins; we would be saying that most penguins are birds and all birds fly, yet no penguins fly. In the second case, we are saying all penguins are birds and most birds fly, resulting in a defeasible argument that penguins fly. This is defeated by the negative, default link between penguins and flying things. If penguins were replaced by an object node, such as TWEETY, both arguments would be of default strength and would rebut each other with no defeat. This would essentially be saying you are fairly certain that TWEETY is a bird and all birds fly, but you are also fairly certain that TWEETY can not fly. In the third example, we are saying penguins are birds and most birds fly, but that penguins do not fly. Clearly the positive argument is defeated, as it would be if penguins were replaced by the object node TWEETY.

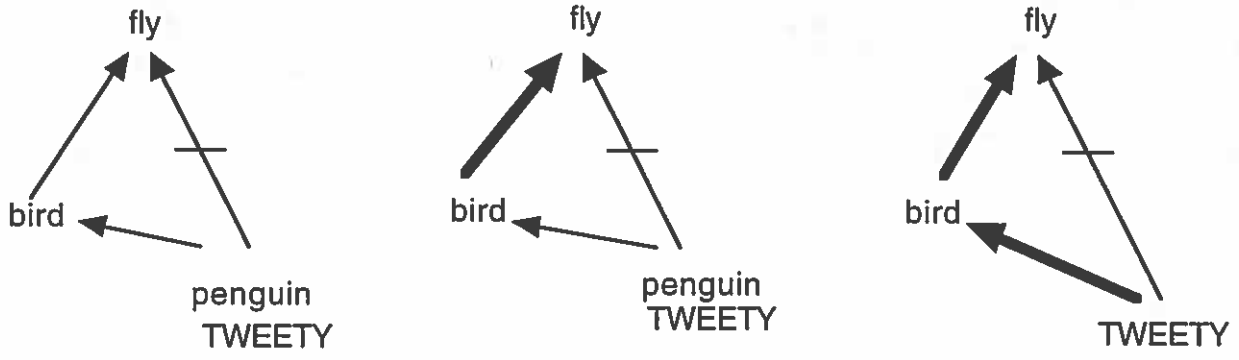

Figure 8.

Figure 3 presents three more "specificity triangles". In the first, where all links are of default strength, the more specific, negative link defeats the other, indirect argument of defeasible strength. In the middle case, both arguments are of default strength. There is no defeat here by the more specific information, only a rebuttal of an equally strong argument through the class of birds. In the third case, the more specific, negative argument of default strength is overruled by the indirect, positive argument that is strong. This last case would be inconsistent if TWEETY were replaced by a class, such as penguins, as discussed in an earlier example above. Two possible triangles have not been shown. The triangle of all strong links is clearly inconsistent. The other case, with the direct, negative link being strong and the other two being default, clearly results in defeat of the indirect argument by the more specific, stronger indication. In these eight triangles, if the link from bird were changed to negative (say to a node labeled swim representing swimming things) and the direct, specific arguments were made positive and indirect arguments negative,

similar results would hold. Thus, our approach provides the full range of specificity results as an outcome of application of our defeat relation based on the comparison of argument strengths.

Next, we consider a slightly more complex situation, given in the network of Figure 9. Here the only argument for the positive claim A~~>p, i..e., P(A, s, r, p), is defeated, based upon the strict, negative link between A and r. As such, the negative claim, having one defendable, argument, i.e., P'(A, q, p), can win arguments up through dialectical validity. If we weaken the negative link between A and r to default strength, then the positive argument is no longer defeated. In this case, both sides of the claim can win scintilla of evidence arguments and only such arguments; each has a single, defendable argument, so neither can win preponderance of evidence arguments. If we now weaken the strength of the link between A and s to default, link A-/->r would defeat the only argument for the positive claim, again creating the original results. If we strengthen the link between A and q to become strict, this argument would dominate the network, allowing the negative claim to win arguments though dialectical validity.

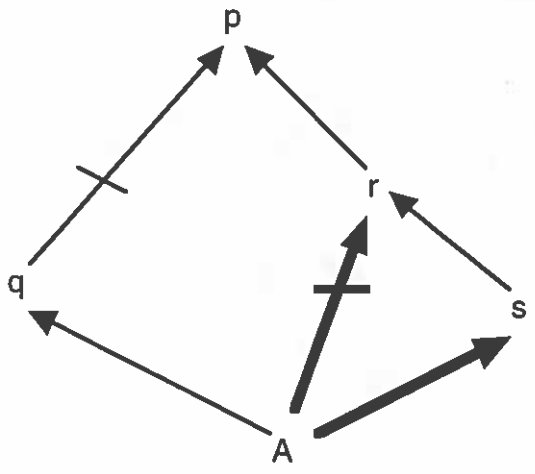

Figure 9.

We make the situation even more complex in Figure 10. Under scintilla of evidence, both positive and negative claims from A to p can be accepted, as each has a defendable, defeasible argument, i.e., P(A, s, r, p) for A~~>p and P'(A, q, p) or P''(A, t, p) for A~/~>p. Under preponderance of evidence, the negative claim remains acceptable as two defeasible arguments exist in support of that claim. Since neither side can defeat the other's rebuttals, these are the only inheritability results. If we make the link between A and q or between A and t strict, then the negative claim can win arguments up through dialectical validity, providing a default argument that defeats the defeasible argument in support of the positive claim. Similarly, if the link between A and s were made strict, the positive claim could win arguments up through dialectical validity.

We can demonstrate the use of candidate negative arguments and modus tollens arguments over strict links in the example of Figure 10, also. If we make class r the goal, we have one positive, defeasible argument for object A being of class r, but three defeasible arguments for the negative side of the claim, including two through class p using modus tollens reasoning over the strict link between r and p. Thus, the negative claim A~/~>r can win arguments up through

preponderance of evidence. The positive claim can only win an argument under scintilla of evidence, as P(A, s, r) is a defendable argument.

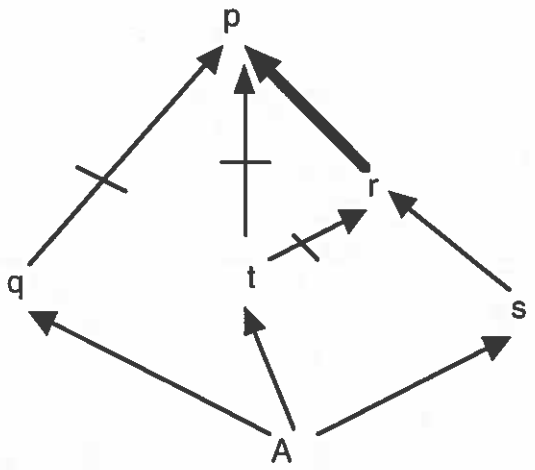

Figure 10.

Next, we present an example that illustrates an important difference between the semantics of our approach and that of several previous approaches relying on argument subsumption. In Figure 11, we see that both positive and negative claims can marshal defendable, defeasible arguments, i.e., P(A, q, s) for A~~>s and P'(A, p, s) for A~/~>s. Thus, both sides can win under scintilla of evidence. Note that the other positive argument P''(A, p, q, s) is defeated by the negative default link between p and s under our scheme. This second positive argument, which includes q as an intermediary between p and s, could be considered a more complete argument (or subsuming explanation) for A~~>s. One could argue that it should defeat the negative argument, which is just the opposite of our conclusion. From that perspective, adding q to the argument is giving further information as to why p is likely to be of kind (or have property) s, and thus can override that link.

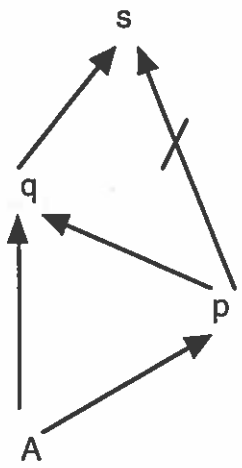

Figure 11.

According to our approach, however, the negative link between p and s in Figure 11 clarifies that while there is a positive, defeasible argument between p and s through q, there actually is a negative, default relation between the classes. This argument is a more specific argument regarding that relationship. From our perspective, defeat of the positive argument involving p does not imply defeat of the positive argument bypassing p and going directly from A to q, which defeat would be another implication of the subsumption framework. As noted above, that approach presumes a different meaning for links of an inheritance network, one that has not enjoyed widespread adoption.

The decision made by other inheritance reasoning schemes not to include the weakening of the inheritance relation over multiple default links and, thus, to view default strength as a transitive property of arguments has caused a number of reported problems in determining acceptable meanings for inheritance networks. In particular, it has led to counterexamples based on particular node labelings that purport to demonstrate network semantics are incorrect. Typically, what is missing from these supposed counterexamples are the direct, default links between nodes that capture implicitly known, unrepresented knowledge about the classes of the network. Under our model, inclusion of these additional links would change the strengths of relevant arguments and, thus, lead to different, correct patterns of inheritability that proposers of the counterexamples note.

As an example, consider the inheritance network of Figure 12 concerning classes of letters and the letter A. With the knowledge given, the positive claims A~~>vowel and A~~>consonant both can win arguments up through dialectical validity, although the argument supporting the vowel claim is of greater strength.

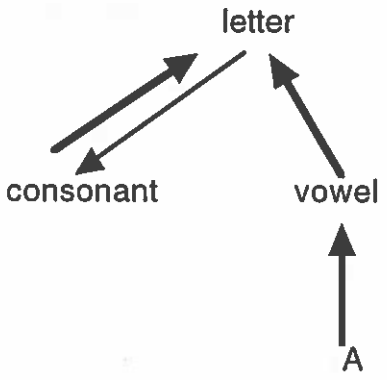

Figure 12.

What is missing from the above example is a representation of the mutual exclusivity of the vowel and consonant classes, as shown in Figure 13. Now, the defeasible argument for the claim A~~>consonant is defeated by the strict, negative arguments P(A, vowel, consonant) and P'(vowel, ø, consonant). As such, there is no confusion as to appropriate conclusions: A is clearly both a vowel and a letter, but not a consonant. The notion of network stability (Horty, 1994), whereby adding default links corresponding to what we consider to be defeasible inheritance arguments does not change network semantics, must be modified. Only if a defeasible

link, i.e., a link type weaker than a default link, were allowed directly in inheritance networks, could we agree with the notion of stability. Otherwise, adding direct default links between nodes connected only by defeasible arguments is an addition of more specific information that should, and in our model does, change network semantics.

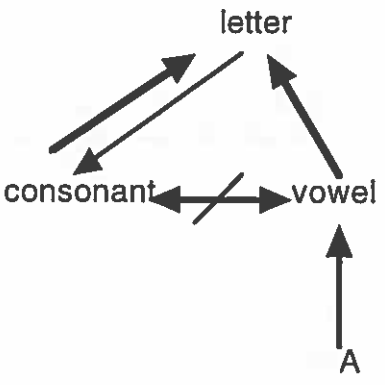

Figure 13.

An Implementation

The implementation of our model in Scheme computes the sets of defendable and justifiable arguments associated with a given inheritance network and determines the sets of claims meeting the differing burdens of proof, based upon those argument sets.

The first step in analyzing an inheritance network is to construct all allowable arguments of the network in accordance with rules R1 and R2, as defined previously. This step is clearly of complexity O(A), where A is the number of allowed arguments in network I. The number of arguments A is in the worst case exponential in N, the number of nodes in I; this occurs, for example, when all links are positive and all node pairs are linked. However, reasonable inheritance hierarchies form a directed lattice of some depth, say D. In the worst case, where the N nodes are equally spread among the D levels, any node at a level can follow any node at prior levels, and all lengths of subarguments are allowable arguments, A is $O(D^2N^2)$.

We then filter out the candidate-negative arguments and compute the subargument relation between remaining arguments. We compute all direct defeats and rebuttals between pairs of arguments, comparing pairs of arguments according to their claims and strengths. Then, we compute the sets all defeats and rebuttals, based upon the subargument relation, recalling that if an argument A directly defeats an argument B and B is a subargument of C, then argument A also defeats C. These algorithms are all implementable with $O(A^2)$ complexity.

We are now in a position to determine the sets of defendable and justifiable arguments. To determine the set of defendable arguments, we first include in the set those arguments that are not attacked by any element of the defeat relation. These arguments are clearly defendable in the given inheritance network I. Then, we remove these defendable arguments, the arguments they defeat

through defeat relation elements having no associated vulnerable arguments, and the defeat relation elements from further consideration. Next, we consider relevant defeat relation elements that have either all of their vulnerable arguments already determined to be defendable or with one vulnerable argument defeated. These defeat relation elements are considered effective (removing the defeated argument from further consideration) or ineffective, respectively; these defeat relation elements are removed from further consideration, as well. We repeat this process until no arguments remain to be considered or until no defeat relations remain to be considered; in the latter case, all remaining arguments are determined to be defendable. This algorithm must converge in a number of cycles less than or equal to the depth of the inheritance network. Since it involves comparing arguments to defeat elements, the time complexity is $O(DA^3)$.

We next determine the set of defendable rebuttals, being those rebuttals between pairs of defendable arguments. Finally, justifiable arguments are those defendable arguments that are not attacked by an element of the defendable rebuttal relation, i.e., they are defendable arguments not rebutted by other defendable arguments. The associated algorithms, involving consideration of arguments with elements of the rebuttal relation, are again implementable in time $O(A^3)$.

Finally, we compute the set of claims supported under each of the three burdens of proof. The claims accepted under scintilla of evidence and dialectical validity correspond directly to claims associated with arguments from the defendable and justifiable argument sets, respectively. The preponderance of evidence results depend upon comparing number of defendable arguments for each pair of conflicting claims in the set of claims accepted under scintilla of evidence.

Appendix I presents three examples, following the steps described above, that illustrate the process of computing the claims supported by an inheritance network I, with or without objects, for the three burdens of proof defined above.

Related Work on Argumentation

As noted earlier, there exists a considerable body of work on inheritance reasoning, much of it based upon the notions of nonmonotonic and default logics (Touretzky, 1986; Reiter, 1980; Ginsberg, 1987; Etherington, 1988). We will not review that work further here; we have related results of that work to outcomes computed under our model in the examples above. We provide an alternative perspective on inheritance reasoning that does not involve computing all consistent extensions or other such logical notions. Our approach focuses on generating and comparing arguments for and against an inheritance claim and making the decision whether to accept the claim in light of the given knowledge and required burden of proof.

As our approach is phrased in the terminology and is considered from the perspective of argumentation, we will review related work in that area. While none of that work has focused directly on inheritance reasoning, the various approaches proposed have considered such reasoning among their example applications. Loui (1987, 1991), Pollock (1987, 1994), and Dung (1995) have proposed specifications of argumentation as a system of formal reasoning and have attempted to characterize its basic elements and relationships. Arguments and the conflict/defeat relations

among them are recognized as being the common, important elements of such approaches.

Loui (1987) considers arguments as directed, acyclic graphs from a set of evidential knowledge EK to a given conclusion C based on a set of defeasible rules R. This perspective can be mapped to ours, as an inheritance network corresponds to a set of rules R, each having a single antecedent, yielding argument paths that end at a goal node C; EK consists of a given object node and its connections to the network. Loui considers defeat as the basic mechanism for resolving conflicts, detected as inconsistencies in argument conclusions. He proposes several bases for defeat: *more evidence*, *specificity*, *directness*, and *preferred premises*. The more evidence criterion, i.e., preferring arguments that use more elements from EK as antecedents, is irrelevant here, as all arguments are paths, each having only one basis in evidence. We could extend Loui's definition of more evidence to mean that more arguments for a side of a claim is preferred, thereby relating it to our decision criterion under preponderance of evidence. We have discussed how issues of defeat by specificity and directness automatically fall out of our scheme of assigning differing argument strengths and determining defeat based on comparison of such strengths. Loui does not propose a scheme for directly assigning strengths to arguments.

We have no mechanisms for preferring certain premises over others in our model, other than as strength of connection in or to the network. Such "extra-argument" considerations could of course be added, but it is unclear as to their determination for inheritance arguments. Prakken and Sartor (1995), considering the domain of legal argumentation, provide reasonable bases for rule priority comparisons in the legal domain. Lex Specialis, Lex Posterior, and Lex Superior are standard bases for resolving conflicts among laws. They provide a way of explicitly reasoning about such rule priorities as an aspect of argumentation. Our only preference relation is argument strength, which seems appropriate for the domain of inheritance reasoning. Loui's discussion of interactions among these factors for defeat is somewhat complex and may remain incomplete in light of the few examples considered. While the defeat relation between pairs of arguments is considered in depth by Loui, little analysis is given to interactions with larger sets of arguments and, in such complex cases, what appropriate outcomes should be.

Pollock (1987, 1994), on the other hand, focuses extensively on the interactions of defeat relations within sets of arguments. He notes the existence of *rebutting defeaters* (direct defeats in our system) and *undercutting defeaters* (subargument defeats in our system). He assigns arguments the strength of their weakest links, leading to two strength possibilities given two link strengths. Strength is not used as a basis for determining defeat in his approach, however. Arguments of equal strength defeat each other, rather than rebut each other, in his model; no distinction is made between these conflicts. This causes his approach certain semantic difficulties; specifically, it forces him to grapple with the issue of *self-defeat*, i.e., where an argument, through its interaction with other arguments, results in its own defeat. As a result, arguments can be in or out at different "levels" of argumentation, as arguments trade their effectiveness as defeaters (being *in*) only to be defeated by another argument at the next level (then being *out*). We encounter this issue when computing the defendable set of arguments; we start with arguments not attacked and place them in as defendable, removing them and arguments they defeat from further consideration.

However, with no cycles in our defeat relations, this process does not cause arguments to alternate as in (defendable) or out, nor must our approach deal explicitly with issues of convergence.

Dung (1995) addresses issues of defeat within sets of arguments and the computation of stable, complete extensions, thereby mixing the argumentation and nonmonotonic reasoning terminology and perspectives. He defines an *argument framework* to consist of a set of arguments and attack relations between them. Sets of *admissible arguments* are defined, analogous to our defendable arguments, i.e. any admissible argument has its attackers attacked by other elements of the set. Since the attack relation can in general include cycles, a fixed point semantics is defined and conditions established that ensure existence of a stable extension. Our inheritance arguments meet the restriction defined there as "limited controversy", thereby guaranteeing that a stable extension exists. Dung does not discuss argument representations for particular applications or the notion of strength for arguments, nor does he distinguish between defeating and rebutting attacks.

Nute (1994) defines a general system of *defeasible logic* that bears significant resemblance to other argumentation approaches, as well as ours. His basic structure is a *proof tree*, consisting of nodes corresponding to well-formed formulae, where each node is labeled by its status with respect to monotonic derivability and demonstrability. His proof trees allow for rules with multiple antecedents; they also intertwine multiple arguments, with node labelings indicating success or failure for sets of opposing arguments. Nute introduces the notion of defeat of defeasible links by strict links but does not consider strength of an overall argument path in determining defeat. He considers defeat to be determined on the basis of a *superiority relation* between links; no preferred premises are proposed as by Loui. He defines, with some complexity, the relationship between rule superiority and rule specificity, limiting this to strict rules and what are termed "defeasible non-suppositions". Again, our use of argument strength as a means of automatically resolving specificity and superiority appears to be a more direct solution for inheritance reasoning than do the proposals made by Nute, though his approach is clearly more general.

Prakken (Prakken, 1993) and Vreeswijk (Vreeswijk, 1993) explore relationships between default logics and formal models of argumentation involving defeasible implications. Again, while their approaches are more general, they do not involve computation and comparison of argument path strengths nor include the notions of burden of proof or vulnerable arguments associated with defeat relations. Their focus is on determining consistent sets of sentences implied by an argument framework. Others (Krause et.al, 1995) recently have addressed issues of confidence measures and the definition of acceptability and associated confidence classes in specifying a so-called logic of argumentation. This approach shares certain commonalities with the work reported here.

Freeman and Farley (Freeman, 1993; Farley and Freeman, 1995), whose work forms the direct background for this present effort, investigate the inclusion of abductive reasoning steps in a dialectical argumentation framework. The two reasoning steps (i.e., asserting the consequent and denying the antecedent) are considered to be fallacies in deductive reasoning. However, they are often appropriate for reasoning when knowledge is incomplete or uncertain (Polya, 1968). In our present inheritance reasoning context, their incorporation would involve extending the indirect argument formation rule R2 to allow positive links to be traversed in the reverse direction (i.e.,

given a-->b and c-->b could result in arguments for b-->a and c-->a) and negative indications to propagate over subsequent positive arcs (i.e., given a-/->b and an argument for b-->c could result in an argument for a-/->c). In their model, such arguments are assigned the least strength of "weak". Thus, they would be defeated by any argument of defeasible strength or better, as formed by the existing rule R2. Such arguments are only effective when no other arguments exist.

Freeman (Freeman, 1993) also formalizes a dialectical argumentation process, whereby two sides alternate in generating arguments for and against a given claim, each in response to arguments posed by the other side. We specialize this model for the domain of inheritance reasoning and demonstrate its connection to our previously defined semantics for burdens of proof in the next section.

Argument as Dialectic Process

Now that we have defined a structure for inheritance arguments and have determined an argument semantics based on the notion of burden of proof, we will discuss a dialectical process whereby we can decide whether to accept a claim regarding property or class y for object X. This dialectical process will represent a procedural semantics that is consistent with the denotational semantics for inheritability defined above.

A dialectical argument has two sides, where *Side-1* argues in favor of an input claim and *Side-2* against that claim, i.e., in support of its negation. The argument process begins with Side-1 attempting to find a grounded argument for the given input claim in terms of the given inheritance network and object description, i.e., a set of links into the inheritance network. If no grounded support can be found, the argumentation process ends in a loss for Side-1; the input claim is not accepted under the given burden of proof. This is consistent with our previously defined semantics; all proof levels require Side-1 to construct at least one grounded, and eventually defendable, argument in support of an input claim.

The two sides alternate taking turns in the role of *active side* of the argument. A side is active until either it succeeds in creating a check condition or it runs out of possible argument moves. A *check condition* for side S of an argument is a situation such that, if the other side can not refute at least one of the relevant arguments of S, the side S wins the argument. In other words, it is a controlling situation, requiring a sufficient response or a concession from the other side. Except for the initial situation, when Side-1 must generate an initial, grounded argument for the claim, the active side is faced with a set of relevant check arguments proposed by the other side. *Check arguments* are those arguments responsible for a side having established a check condition.

The active side tries to apply one of several possible argument moves to recapture the check condition. These *argument moves* are, as follows: defeat-check-argument, rebut-check-argument, and generate-new-supporting-argument. The active side can apply one of two primitive functions to search inheritance network I for relevant arguments. The first is *find-arguments (x, g, s, I)*, which searches for argument paths from node x to node g of sign s in inheritance network I. The

function returns (first of) a list of argument paths sorted in decreasing order of argument strength or returns an empty list indicating no such paths exist. The second function is *find-conflicting-arguments(A, I)*, which similarly finds arguments that conflict with the given argument A. This set is equivalent to the union of all conflicting arguments for each relevant pair of nodes in argument A. Assuming that the given knowledge is consistent, only pairs of nodes that isolate subarguments of defeasible or default strength need be considered. This second function can be implemented by calls to the function *find-arguments* with parameters being relevant pairs of nodes, x and g, from argument A and the complement of the sign of the subargument between x and g in argument A.

Whether an argument is found that is adequate to generate a check condition for the active side depends upon the burden of proof it faces. Under a burden of proof of dialectical validity, Side-2 can consider both defeat-check-argument and rebut-check-argument moves in response to Side-1's arguments supporting the given claim. If Side-2 finds a sufficient check argument, Side-1 must either defeat Side-2's response or propose a completely new argument for its claim; otherwise, it must concede the argument. Side-2 can continue throwing up arguments that conflict with any check argument currently proposed by Side-1 in support of the claim. On the other hand, Side-1 must defeat any such argument or propose a new argument altogether, if it is to prevail under this burden of proof. If Side-2 can not find a conflicting argument, it must concede. These rules for the two sides of the argument are clearly consistent with our earlier definition of dialectical validity. A claim is accepted if a justifiable argument can be found.

When the burden of proof is merely scintilla of evidence, Side-2 can only consider the defeat-check-argument move when responding to Side-1's check condition. Side-1 need not defeat rebuttals to win an argument under this burden of proof; it must merely defend some argument against defeat. If Side-2 does defeat an argument proposed by Side-1, Side-1 can either try to defeat the defeating argument or abandon that argument in favor of another that supports the input claim, i.e., apply the generate-new-supporting-argument move. If Side-2 fails to defeat a check argument of Side-1, it must concede; if Side-1 can not defeat Side-2's check argument or can not find an alternative argument for the claim, it must concede. These requirements are consistent with our earlier, denotational definition of this burden of proof. A claim is accepted if supported by a defendable argument.

Finally, if the burden of proof is preponderance of evidence, Side-2 must generate an argument that either defeats of directly conflicts with that proposed by Side-1. If it can, Side-1 must in turn either defeat one of Side-2's check arguments or generate another grounded argument in favor of the claim. As long as Side-1 has more undefeated arguments, it will prevail. If either Side-1 or Side-2 fails to find a rebutting or defeating argument, it must concede the argument. Once again, the argument requirements match our earlier semantic definition of the proof level.

We can characterize the three different burdens of proof in terms of where they place an associated "burden of defeat", i.e., which side must defeat the other's arguments during the argument process. In the case of scintilla of evidence, the burden of defeat is on Side-2; under a burden of proof of dialectical validity, the burden of defeat is on Side-1. Under preponderance of evidence, neither side assumes a burden of defeat; each side can choose to defeat the other's

arguments or to offer yet another argument for or against the given claim of sufficient strength.

The above dialectical argumentation process is readily implementable in terms of our implemented functions demonstrated in Appendix I. First, we compute all arguments and all subargument, rebuttal, and defeat relations for a network and index them by start and finish nodes. Given an object description, these sets and indices can be extended to include arguments starting with the object node, forming a *compiled network.* Then, given a claim regarding the object and a burden of proof, the three argument moves and the functions *find-arguments* and *find-conflicting-arguments* can use the previously compiled network to conduct the dialectic process according to the rules outlined above.

Conclusion

In this paper, we have proposed a rethinking of inheritance reasoning and its semantics based upon arguments constructed directly in terms of the structure of mixed inheritance networks. The distinguishing elements of our approach are (i) treating inheritance reasoning as a form of dialectical argumentation; (ii) eliminating transitivity of the default relation, thereby weakening the strength of default arguments to become defeasible arguments when extended; and (iii) introducing burden of proof as means for allocating risk and determining acceptability of inheritability claims. We have implemented our notions of inheritance argumentation, with algorithms requiring time and space at most polynomial in the number of allowable arguments in an inheritance network. While the number of arguments can be exponential in the worst case, for most inheritance networks that are directed acyclic graphs this number will be polynomial.

Our approach generalizes many of the positive properties of direct, network-based methods of inheritance reasoning that have been proposed previously and clarifies or eliminates other issues that have caused this area of inquiry difficulty in the past. In particular, issues concerning the nature of credulous and skeptical systems, e.g., which is the most appropriate and how to define them, are replaced by a defeat relation defined in terms of relative strengths among conflicting argument and a burden of proof defined in terms of sets of defendable or justifiable arguments. The ability to adjust the burden of proof as a parameter that reflects risk acceptability in a particular situation, rather than attributing it as a fixed property (either skeptical or credulous) of a reasoning system, provides the flexibility of control required by systems when making decisions regarding inheritance semantics in real-world situations.

References

Dung, P.M. 1995. "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games", *Artificial Intelligence*, 77(2), 321-358.

Etherington, D. 1988. *Reasoning with Incomplete Knowledge*, Morgan Kaufmann: Menlo Park, CA.

Fahlman, S. 1979. *NETL: A System for Representing and Using Real-world Knowledge*, Morgan Kaufmann: Menlo Park, CA.

Fahlman, S., Touretzky, D. and van Roggen, W. 1981. "Cancellation in a parallel semantic network", in *Proceedings of IJCAI-81*, 257-263.

Farley, A.M. and Freeman, K. 1995. "Burden of proof in legal argumentation", in *Proceedings of Fifth International Conference on Artificial Intelligence and Law*, 156-163.

Freeman, K. *A Computational Model of Dialectical Argumentation*, Ph.D. Thesis, Computer and Information Science Department, University of Oregon, 1993.

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

Horty, J.F. 1994. "Some direct theories of nonmonotonic inheritance", in Gabbay, D.M., Hogger, C.J, and Robinson, J.A. (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*, Oxford Press : New York, 111-188.

Horty, J.F. and Thomason, R. 1988. "Mixing strict and defeasible inheritance", in *Proceedings AAAI-88*, 427-432.

Krause, P., Ambler, S., Elvang-Goransson, M. and Fox, J. 1995. "A logic of argumentation for reasoning under uncertainty", *Computational Intelligence*, 110-131.

Loui, R. "Defeat among arguments: a system of defeasible inference", 1987. *Computational Intelligence*, 3, 100-106.

Loui, R. 1991. "Argument and belief: wehre we stand in the Keynesian tradition", *Minds and Machines*, 1, 357-366.

Nute, D. 1994. "Defeasible logic", in Gabbay, D.M., Hogger, C.J, and Robinson, J.A. (eds). *Handbook of Logic in Artificial Intelligence and Logic Programming*, Oxford Press: New York, 353-395, 1994.

Pollock, J. 1987. "Defeasible reasoning", *Cognitive Science*, 11, 481-518.

Pollock, J. 1994. "Justification and defeat", *Artificial Intelligence*, 67, 377-407.

Polya, G. 1968. *Mathematics and plausible reasoning* (2nd ed.), Princeton University Press: Princeton, NJ.

Prakken, H. 1993. "An argumentation framework in default logic", Annals of Mathematics and Artificial Intelligence (9), 93-132.

Prakken, H. and Sartor, G. 1995. "On the relation between legal language and legal argument: assumptions, applicability, and dynamic priorities", in *Proceedings of Fifth International Conference on Artificial Intelligence and Law*, 1-10.

Quillian, J.R. 1968. "Semantic memory", in M. Minsky(ed.), *Semantic Information Processing*, 227-270.

Reiter, R. 1980. "A logic for default reasoning", *Artificial Intelligence*, 13, 81-132.

Sandewall, E. 1986. "Nonmonotonic inference rules for multiple inheritance with exceptions", in *Proceedings of the IEEE*, 74: 1345-1353.

Sartor, G. 1993. "A simple model for nonmonotonic and adversarial legal reasoning", in *Proceedings of Fourth International Conference on Artificial Intelligence and Law*, 192-201.

Sowa, J.F. 1984. *Conceptual Structures*, Addision-Wesley: Reading, MA.

Sowa, J.F. (ed.) 1992. *Principles of Semantic Networks*, Morgan-Kaufman: Menlo Park, CA.

Touretzky, D. 1986. *The Mathematics of Inheritance Systems*, Pitman, London.

Touretzky, D., Horty, J.F., and Thomason, R. 1987. "A clash of intuitions: the current state of nonmonotonic multiple inheritance systems", in *Proceedings of IJCAI-87*, 476-482.

Vreeswijk, G. 1993. *Studies in Defeasible Argumentation*, Ph.D. Dissertation, Vrije Universiteit, Amsterdam, Netherlands.

Woods, W. 1975. "What's in a link: foundations of semantic networks", in Bobrow, D. and Collins, A. (eds.) *Representation and Understanding: Studies in Cognitive Science*, 35-82, Academic Press: New York.

Appendix I

As our first example, consider the network of Figure A1. In this example, those born in Pennsylvania (bp) are clearly American born (ab), while almost all german speakers (gs) are not born in America. Pennsylania Dutch speakers (pds) are for the most part born in Pennsylvania, and, by definition, are german speakers. We know of two people, HERMANN and ADELLE, who are both Pennsylvania Dutch Speakers, while ADELLE is also known not to be born in Pennsylvania.

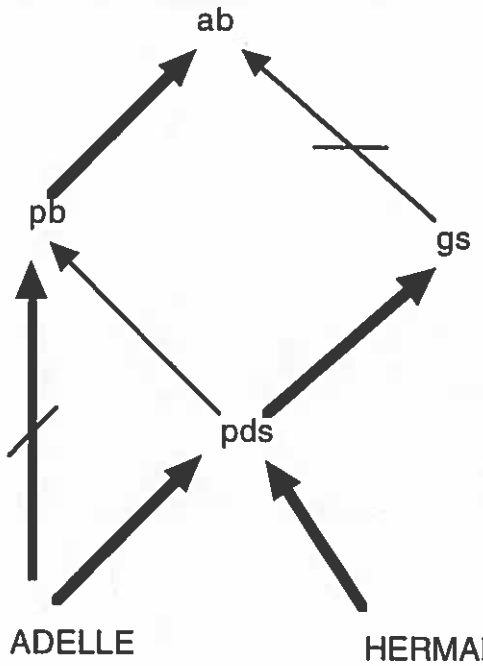

Figure A1.

This network is described to our implementation as a list of arcs, as follows:

```
>>> (define netpd-arcs
  (list (make-arc 'pb 'ab '+ 'strict)
        (make-arc 'pds 'pb '+ 'default)
        (make-arc 'pds 'gs '+ 'strict)
        (make-arc 'gs 'ab '- 'default)
        (make-arc 'HERMANN 'pds '+ 'strict)
        (make-arc 'ADELLE 'pb '- 'strict)
        (make-arc 'ADELLE 'pds '+ 'strict)))
netpd-arcs
>>> (define netpd (make-net netpd-arcs) '(HERMANN ADELLE))  ;;  Hermann and Adelle are objects
netpd
```

The first step in analyzing the network is to construct all allowable arguments from the network, in accordance with rules R1 and R2 defined in the paper, as follows:

```
>>> (define netpd-all-args (all-arguments netpd))
netpd-all-args
```

```
>>> (pretty-print  netpd-all-args)
((argument arg52 (claim adelle pds +) strict (adelle pds)
          ((arc adelle pds + strict)))
 (argument arg53 (claim adelle pb -) strict (adelle pb)
          ((arc adelle pb - strict)))
 (argument arg54 (claim hermann pds +) strict (hermann pds)
          ((arc hermann pds + strict)))
 (argument arg74 (claim hermann ab -) defeasible (hermann pds gs ab)
          ((arc hermann pds + strict) (arc pds gs + strict) (arc gs ab - default)))
 (argument arg73 (claim adelle ab -) defeasible (adelle pds gs ab)
          ((arc adelle pds + strict) (arc pds gs + strict) (arc gs ab - default)))
 (argument arg72 (claim pds ab -) defeasible (pds gs ab)
          ((arc pds gs + strict) (arc gs ab - default)))
 (argument arg55 (claim gs ab -) default (gs ab)
          ((arc gs ab - default)))
 (argument arg56 (claim gs pds -) cand-neg (gs pds)
          ((arc pds gs + strict)))
 (argument arg71 (claim hermann gs +) strict (hermann pds gs)
          ((arc hermann pds + strict) (arc pds gs + strict)))
 (argument arg70 (claim adelle gs +) strict (adelle pds gs)
          ((arc adelle pds + strict) (arc pds gs + strict)))
 (argument arg57 (claim pds gs +) strict (pds gs)
          ((arc pds gs + strict)))
 (argument arg69 (claim hermann pb +) default (hermann pds pb)
          ((arc hermann pds + strict) (arc pds pb + default)))
 (argument arg68 (claim adelle pb +) default (adelle pds pb)
          ((arc adelle pds + strict) (arc pds pb + default)))
 (argument arg58 (claim pds pb +) default (pds pb)
          ((arc pds pb + default)))
 (argument arg67 (claim hermann pb -) defeasible (hermann pds gs ab pb)
          ((arc hermann pds + strict) (arc pds gs + strict) (arc gs ab - default) (arc pb ab + strict)))
 (argument arg66 (claim adelle pb -) defeasible (adelle pds gs ab pb)
          ((arc adelle pds + strict) (arc pds gs + strict) (arc gs ab - default) (arc pb ab + strict)))
 (argument arg65 (claim pds pb -) defeasible (pds gs ab pb)
          ((arc pds gs + strict) (arc gs ab - default) (arc pb ab + strict)))
 (argument arg64 (claim gs pb -) default (gs ab pb)
          ((arc gs ab - default) (arc pb ab + strict)))
 (argument arg59 (claim ab pb -) cand-neg (ab pb)
          ((arc pb ab + strict)))
 (argument arg63 (claim hermann ab +) default (hermann pds pb ab)
          ((arc hermann pds + strict) (arc pds pb + default) (arc pb ab + strict)))
 (argument arg62 (claim adelle ab +) default (adelle pds pb ab)
          ((arc adelle pds + strict) (arc pds pb + default) (arc pb ab + strict)))
 (argument arg61 (claim pds ab +) default (pds pb ab)
          ((arc pds pb + default) (arc pb ab + strict)))
 (argument arg60 (claim pb ab +) strict (pb ab) ((arc pb ab + strict))))
```

We filter out the candidate-negative arguments and compute the subargument relation between the remaining arguments, as follows:

```
>>> (define netpd-cand-neg-args (get-cand-neg-args netpd-all-args))
netpd-cand-neg-args
>>> (display (get-arg-names netpd-cand-neg-args))
(arg56  arg59)
```

```
>>> (define netpd-args (filter-cand-neg-args netpd-all-args))
netpd-args
>>> (define netpd-subs (subarguments netpd-args))
netpd-subs
>>>  (display netpd-subs)
((subargument arg52 arg73) (subargument arg52 arg70) (subargument arg52 arg68)
 (subargument arg52 arg66) (subargument arg52 arg62) (subargument arg54 arg74)
 (subargument arg54 arg71) (subargument arg54 arg69) (subargument arg54 arg67)
 (subargument arg54 arg63) (subargument arg74 arg67) (subargument arg73 arg66)
 (subargument arg72 arg74) (subargument arg72 arg73) (subargument arg72 arg67)
 (subargument arg72 arg66) (subargument arg72 arg65) (subargument arg55 arg74)
 (subargument arg55 arg73) (subargument arg55 arg72) (subargument arg55 arg67)
 (subargument arg55 arg66) (subargument arg55 arg65) (subargument arg55 arg64)
 (subargument arg71 arg74) (subargument arg71 arg67) (subargument arg70 arg73)
 (subargument arg70 arg66) (subargument arg57 arg74) (subargument arg57 arg73)
 (subargument arg57 arg72) (subargument arg57 arg71) (subargument arg57 arg70)
 (subargument arg57 arg67) (subargument arg57 arg66) (subargument arg57 arg65)
 (subargument arg69 arg63) (subargument arg68 arg62) (subargument arg58 arg69)
 (subargument arg58 arg68) (subargument arg58 arg63) (subargument arg58 arg62)
 (subargument arg58 arg61) (subargument arg65 arg67) (subargument arg65 arg66)
 (subargument arg64 arg67) (subargument arg64 arg66) (subargument arg64 arg65)
 (subargument arg61 arg63) (subargument arg61 arg62) (subargument arg60 arg63)
 (subargument arg60 arg62) (subargument arg60 arg61))
```

We compute all direct defeats and rebuttals between pairs of arguments, comparing their claims and strengths. Then, we compute all defeats and rebuttals based upon the subargument relation, recalling that if argument A directly defeats (rebuts) argument B and B is a subargument of C, then argument A also defeats (rebuts) C. We include as part of each relation element a list of associated vulnerable arguments. No vulnerable arguments are associated with direct defeats.

```
>>>(define netpd-ddrs (direct-defeaters-rebutters netpd-args))
netpd-ddrs
>>> (display netpd-ddrs)
((defeats arg53 arg68 ()) (defeats arg69 arg67 ()) (defeats arg68 arg66 ())
 (defeats arg58 arg65 ()) (defeats arg63 arg74 ()) (defeats arg62 arg73 ())
 (defeats arg61 arg72 ()))
>>> (define netpd-adrs (all-defeats-rebuttals netpd-ddrs netpd-subs netpd-args))
netpd-adrs
>>> (define netpd-ads (select-defeats netpd-adrs))
netpd-ads
>>> (display netpd-ads)
((defeats arg53 arg68 ()) (defeats arg53 arg62 ()) (defeats arg69 arg67 ())
 (defeats arg68 arg66 ()) (defeats arg58 arg65 ()) (defeats arg58 arg67 ())
 (defeats arg58 arg66 ()) (defeats arg63 arg74 ()) (defeats arg63 arg67 ())
 (defeats arg62 arg73 ()) (defeats arg62 arg66 ()) (defeats arg61 arg72 ())
 (defeats arg61 arg74 (arg69)) (defeats arg61 arg73 (arg68)) (defeats arg61 arg67 (arg69))
 (defeats arg61 arg66 (arg68)) (defeats arg61 arg65 ()))
>>> (define netpd-ars (select-rebuttals netpd-adrs))
netpd-ars
>>> (display netpd-ars)
()
```

We can now determine the sets of defendable and justifiable arguments, as described in the paper. In this case, the results are, as follows:

```
>>> (define netpd-das (defendable-arguments netpd-ads netpd-args))
netpd-das
>>> (display (get-arg-names netpd-das))
(arg73 arg52 arg53 arg54 arg55 arg71 arg70 arg57 arg69 arg58 arg64 arg63 arg61 arg60)
>>> (define netpd-defrs (defendable-rebuttals netpd-ars netpd-das))
netpd-defrs
>>>  (define netpd-jas (justifiable-arguments netpd-das netpd-defrs))
netpd-jas
>>> (define netpd-sec (se-claims netpd-das netpd-jas))
netpd-sec
>>> (define net2-dvc (dv-claims netpd-das netpd-jas))
net2-dvc
>>> (define net2-pec (pe-claims netpd-das netpd-jas))
net2-pec
>>> (display netpd-sec)
((claim adelle ab -) (claim adelle pds +) (claim adelle pb -) (claim hermann pds +)
 (claim gs ab -) (claim hermann gs +) (claim adelle gs +) (claim pds gs +)
 (claim hermann pb +) (claim pds pb +) (claim gs pb -) (claim hermann ab +)
 (claim pds ab +) (claim pb ab +))
```

In this example, there are no defendable rebuttals; thus, the set of justifiable arguments is the same as the set of defendable arguments. We then compute the set of claims supported under burden of proof of scintilla of evidence, being those associated with defendable arguments. Of particular interest, as per our discussion regarding vulnerability of defeat relations, is that claims (claim adelle pb -) and (claim adelle ab -) are defendable (and justifiable) for ADELLE, while claims (claim hermann pb +) and (claim hermann ab +) are instead supported for HERMANN.

We next look at results generated for the inheritance network of Figure 4 of the paper, which results in differing sets of claims being accepted at each burden of proof.

```
>>> (define net2-arcs
  (list (make-arc 'a 'b '+ 'default)
        (make-arc 'a 'c '+ 'default)
        (make-arc 'a 'd '+ 'default)
        (make-arc 'b 'f '- 'default)
        (make-arc 'c 'f '+ 'default)
        (make-arc 'd 'f '- 'default)
        (make-arc 'f 'g '+ 'default)))
net2-arcs
>>> (define net2 (make-net net2-arcs '())) ;; no objects in the network
net2
>>> (define net2-all-args (all-arguments net2))
net2-all-args
>>>  (pretty-print  net2-all-args)
((argument arg68 (claim a g +) defeasible (a c f g)
          ((arc a c + default) (arc c f + default) (arc f g + default)))
 (argument arg67 (claim c g +) defeasible (c f g)
          ((arc c f + default) (arc f g + default)))
```

```
(argument arg57 (claim f g +) default (f g) ((arc f g + default)))
(argument arg66 (claim a f -) defeasible (a d f)
        ((arc a d + default) (arc d f - default)))
(argument arg58 (claim d f -) default (d f) ((arc d f - default)))
(argument arg65 (claim a f +) defeasible (a c f)
        ((arc a c + default) (arc c f + default)))
(argument arg59 (claim c f +) default (c f) ((arc c f + default)))
(argument arg64 (claim a f -) defeasible (a b f)
        ((arc a b + default) (arc b f - default)))
(argument arg60 (claim b f -) default (b f) ((arc b f - default)))
(argument arg61 (claim a d +) default (a d) ((arc a d + default)))
(argument arg62 (claim a c +) default (a c) ((arc a c + default)))
(argument arg63 (claim a b +) default (a b) ((arc a b + default))))
```

This time there are no defeats, only rebuttals, between arguments.

```
>>> (display net2-adrs)
((rebuts arg66 arg65 ()) (rebuts arg66 arg68 ()) (rebuts arg65 arg66 ())
  (rebuts arg65 arg64 ()) (rebuts arg64 arg65 ()) (rebuts arg64 arg68 ()))
```

While all claims can survive under scintilla of evidence, the presence of differing numbers of arguments for various claims and of defendable rebuttals serves to eliminate several claims at higher burdens of proof.

```
>>> (pretty-print net2-sec)
((claim a g +) (claim c g +) (claim f g +) (claim d f -)
  (claim a f +) (claim c f +) (claim a f -) (claim b f -)
  (claim a d +) (claim a c +) (claim a b +))
>>> (pretty-print net2-pec)  ;; does not include (claim a f +)
((claim c g +) (claim f g +) (claim d f -) (claim c f +)
  (claim b f -) (claim a d +) (claim a c +) (claim a b +)
  (claim a g +) (claim a f -))
>>> (pretty-print net2-dvc)  ;; does no include (claim a f -) or (claim a g +)
((claim c g +) (claim f g +) (claim d f -) (claim c f +)
  (claim b f -) (claim a d +) (claim a c +) (claim a b +))
```

We consider one more example, this time involving defeat in the context of a cascade of vulnerable arguments, as shown in Figure A.2. In particular, consider the accepted relationship between b, c, d, and e with respect to a. We see claim (b a +) is supported by a single, default argument and is not challenged. The defeasible argument from c to a through b is defeated by the negative argument through g, so claim (c a -) is has a defendable argument. Considering d, the defeasible argument through c and b is defeated by the negative argument from c through g, but the defeat is vulnerable with respect to arguments regarding g. In fact, the argument though h defeats the vulnerable argument associated with the defeat; so, claim (d a +) has a defendable argument. Finally, the default arc from e to h defeats a vulnerable argument associated with the defeat of the negative argument through g; thus, claim (e a -) has a defendable argument.

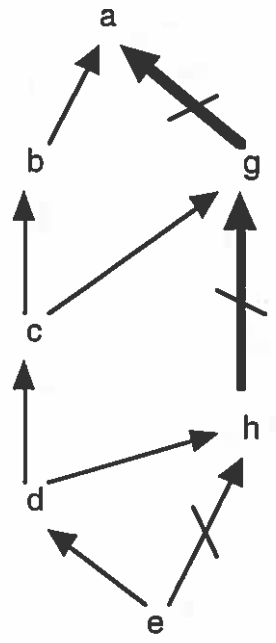

Figure A.2

We consider important elements of the trace determining the claims satifying burdens of proof of scintilla of evidence and dialectical validity, as follows.

```
>>> (define net-new-arcs
  (list (make-arc 'b 'a '+ 'default)
        (make-arc 'c 'b '+ 'default)
        (make-arc 'd 'c '+ 'default)
        (make-arc 'c 'g '+ 'default)
        (make-arc 'g 'a '- 'strict)
        (make-arc 'd 'h '+ 'default)
        (make-arc 'h 'g '- 'strict)
        (make-arc 'e 'd '+ 'default)
        (make-arc 'e 'h '- 'default)))
net-new-arcs
>>>  (define net-new (make-net net-new-arcs) '())  ;; no objects in the network
net-new
>>> (define net-new-all-args (all-arguments net-new))
net-new-all-args
>>> (define net-new-cand-neg-args (get-cand-neg-args net-new-all-args))
net-new-cand-neg-args
>>> (define net-new-args (filter-cand-neg-args net-new-all-args))
net-new-args

>>> (pretty-print net-new-args)
((argument arg26 (claim e h -) default (e h) ((arc e h - default)))
 (argument arg27 (claim e d +) default (e d) ((arc e d + default)))
 (argument arg57 (claim e h -) defeasible (e d c g h)
          ((arc e d + default) (arc d c + default) (arc c g + default) (arc g h - strict)))
 (argument arg56 (claim d h -) defeasible (d c g h)
         ((arc d c + default) (arc c g + default) (arc g h - strict)))
```

```
(argument arg55 (claim c h -) default (c g h)
        ((arc c g + default) (arc g h - strict)))
(argument arg28 (claim g h -) strict (g h) ((arc g h - strict)))
(argument arg54 (claim e g -) defeasible (e d h g) ((arc e d + default)
         (arc d h + default) (arc h g - strict)))
(argument arg53 (claim d g -) default (d h g)
        ((arc d h + default) (arc h g - strict)))
(argument arg29 (claim h g -) strict (h g) ((arc h g - strict)))
(argument arg52 (claim e h +) defeasible (e d h)
       ((arc e d + default) (arc d h + default)))
(argument arg30 (claim d h +) default (d h) ((arc d h + default)))
(argument arg51 (claim e g -) defeasible (e d c b a g)
        ((arc e d + default) (arc d c + default) (arc c b + default) (arc b a + default) (arc a g - strict)))
(argument arg50 (claim d g -) defeasible (d c b a g)
        ((arc d c + default) (arc c b + default) (arc b a + default) (arc a g - strict)))
(argument arg49 (claim c g -) defeasible (c b a g)
        ((arc c b + default) (arc b a + default) (arc a g - strict)))
(argument arg48 (claim b g -) default (b a g)
        ((arc b a + default) (arc a g - strict)))
(argument arg31 (claim a g -) strict (a g) ((arc a g - strict)))
(argument arg47 (claim e a -) defeasible (e d c g a)
        ((arc e d + default) (arc d c + default) (arc c g + default) (arc g a - strict)))
(argument arg46 (claim d a -) defeasible (d c g a)
        ((arc d c + default) (arc c g + default) (arc g a - strict)))
(argument arg45 (claim c a -) default (c g a)
        ((arc c g + default) (arc g a - strict)))
(argument arg32 (claim g a -) strict (g a) ((arc g a - strict)))
(argument arg44 (claim e g +) defeasible (e d c g)
        ((arc e d + default) (arc d c + default) (arc c g + default)))
(argument arg43 (claim d g +) defeasible (d c g)
        ((arc d c + default) (arc c g + default)))
(argument arg33 (claim c g +) default (c g) ((arc c g + default)))
(argument arg42 (claim e c +) defeasible (e d c)
        ((arc e d + default) (arc d c + default)))
(argument arg34 (claim d c +) default (d c) ((arc d c + default)))
(argument arg41 (claim e b +) defeasible (e d c b)
        ((arc e d + default) (arc d c + default) (arc c b + default)))
(argument arg40 (claim d b +) defeasible (d c b)
        ((arc d c + default) (arc c b + default)))
(argument arg35 (claim c b +) default (c b) ((arc c b + default)))
(argument arg39 (claim e a +) defeasible (e d c b a)
        ((arc e d + default) (arc d c + default) (arc c b + default) (arc b a + default)))
(argument arg38 (claim d a +) defeasible (d c b a)
        ((arc d c + default) (arc c b + default) (arc b a + default)))
(argument arg37 (claim c a +) defeasible (c b a)
        ((arc c b + default) (arc b a + default)))
(argument arg36 (claim b a +) default (b a) ((arc b a + default))))
>>> (define net-new-subs (subarguments net-new-args))
net-new-subs
>>> (define net-new-ddrs (direct-defeaters-rebutters net-new-args))
net-new-ddrs
>>>  (define net-new-adrs (all-defeats-rebuttals net-new-ddrs net-new-subs net-new-args))
net-new-adrs
>>> (define net-new-ads (select-defeats net-new-adrs))
```

```
net-new-ads
>>>  (display net-new-ads)
((defeats arg26 arg52 ()) (defeats arg26 arg54 ()) (defeats arg53 arg43 ())
 (defeats arg53 arg57 (arg52)) (defeats arg53 arg56 ()) (defeats arg53 arg47 (arg52))
 (defeats arg53 arg46 ()) (defeats arg53 arg44 (arg52)) (defeats arg30 arg56 ())
 (defeats arg30 arg57 ()) (defeats arg45 arg37 ()) (defeats arg45 arg51 (arg44))
 (defeats arg45 arg50 (arg43)) (defeats arg45 arg49 ()) (defeats arg45 arg39 (arg44))
 (defeats arg45 arg38 (arg43)) (defeats arg33 arg49 ()) (defeats arg33 arg51 ())
 (defeats arg33 arg50 ()))
>>> (define net-new-ars (select-rebuttals net-new-adrs))
net-new-ars
>>> (define net-new-das (defendable-arguments net-new-ads net-new-args))
net-new-das
>>>  (define net-new-defrs (defendable-rebuttals net-new-ars net-new-das))
net-new-defrs
>>> (define net-new-jas (justifiable-arguments net-new-das net-new-defrs))
net-new-jas
>>> (define net-new-sec (se-claims net-new-das net-new-jas))
net-new-sec
>>> (pretty-print net-new-sec)
((claim e a -) (claim e g +) (claim d a +)
 (claim e h -) (claim e d +) (claim c h -) (claim g h -)
 (claim d g -) (claim h g -) (claim d h +) (claim b g -)
 (claim a g -) (claim c a -) (claim g a -) (claim c g +)
 (claim e c +) (claim d c +) (claim e b +) (claim d b +)
 (claim c b +) (claim b a +))
```

Here, we see the prefix inclusion property upheld, but the suffix inclusion property is denied repeatedly. The claims (claim b a +) and (claim d a +) are accepted at the level of scintilla of evidence, while claims (claim c a -) and (claim e a -) are similarly accepted. If we determine the set of justifiable arguments and their associated claims under dialectical validity, we find the claim (claim e a -) does not survive; its support is rebutted by the defendable argument for (claim d a +).

```
>>> net-new-defrs
((rebuts arg38 arg47 (arg41 arg42)))
>>> (define net-new-dvc (dv-claims net-new-das net-new-jas))
(pretty-print  net-new-dvc)
>>> (define net-new-pec (pe-claims net-new-das net-new-jas))
net-new-pec
>>> (pretty-print net-new-dvc)  ;; all but (e a -) as (d a +) is a (subargument) rebuttal
((claim e g +) (claim d a +) (claim e h -) (claim e d +)
 (claim c h -) (claim g h -) (claim d g -) (claim h g -)
 (claim d h +) (claim b g -) (claim a g -) (claim c a -)
 (claim g a -) (claim c g +) (claim e c +) (claim d c +)
 (claim e b +)  (claim d b +) (claim c b +) (claim b a +))
```