Semantically Sound Inheritance for a Formally Defined Frame Language with Defaults

Robert Nado and Richard Fikes
IntelliCorp
1975 El Camino Real West
Mountain View, California 94040-2216

Abstract

Most frame languages either are glaringly deficient in their treatment of default information or do not represent it at all. This paper presents a formal description of a frame language that provides semantically sound facilities for representing default information and an efficient serial algorithm for inheriting default information down class-subclass and class-member hierarchies constructed in that language. We present the inheritance algorithm in two forms. In the first form, the algorithm provides justifications to a TMS, which then manages the inherited information. In the second form, the algorithm performs its own, special-purpose truth maintenance and therefore is useable in a system that does not include a general-purpose TMS.[1]

I. Introduction

The common-sense reasoning required in many knowledge system applications relies heavily on the ability to use general information that is subject to exceptions: what has been called prototypic or default information. Although frame-based representation languages have become increasingly popular for expressing the domain-specific information on which the functionality of knowledge systems is based [Fikes and Kehler, 1985], most such languages either are glaringly deficient in their treatment of default information (as argued, for example, in [Brachman, 1985] and [Touretzky, 1984]) or do not represent it at all (e.g., KL-ONE [Brachman and Schmolze, 1985] and KRYPTON [Brachman *et al.*, 1983]). Thus, an important step in the advancement of knowledge system technology is the development of a frame language that provides semantically sound facilities for representing and efficiently processing default information. This paper presents a formal description of such a frame language (based on the frame language in the KEE™ system[2]) and an efficient serial algorithm for inheriting default information down class-subclass and class-member hierarchies constructed in that language. The language has been implemented at IntelliCorp in a system called OPUS.

As observed by Touretzky [Touretzky, 1986], the "shortest-path" ordering of defaults used by most inheritance systems (e.g., FRL [Roberts and Goldstein, 1977] and NETL [Fahlman, 1979]), does not always successfully provide the desired preference of more specific defaults over less specific defaults. Problems arise in some cases of multiple inheritance, where nodes are allowed to have more than one parent link. An example, adapted from Touretzky, is depicted in Figure 1. The typical inheritance algorithm correctly prefers White over Grey as a default color for a royal elephant, because the default from RoyalElephants has a "shorter path" than the default from Elephants. However, in the situation shown in the figure, Clyde has a redundant class membership link to Elephants. Clyde, then, inherits both the default White from RoyalElephants and the default Grey from Elephants *along paths of equal length*. Thus, shortest-path algorithms are not sufficient to correctly handle this situation.[3] This, and other shortcomings of existing algorithms are overcome in the OPUS algorithm presented here.

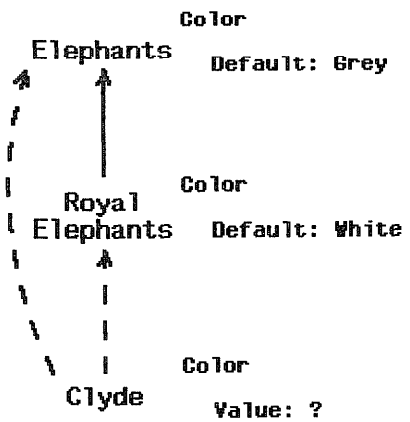

Figure 1: A Problem with the "Shortest Path" Ordering

An additional motivation for this work is to enable "truth maintenance" (or, "reason maintenance" as it is sometimes called) capabilities to be incorporated into frame-based representation systems. Truth maintenance algorithms provide an automatic means of managing derived results as changes are made in a model [Doyle, 1979]. In addition, a truth maintenance system (TMS) can be used as the basis for a context mechanism that enables a frame system to model and compare multiple hypothetical situations (as was done, for example, in the KEEworlds™ facility [Morris and Nado, 1986]).

[1]This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. F30602 85 C 0065. The views and conclusions reported here are those of the authors and should not be construed as representing the official position or policy of DARPA or the U.S. government.

[2]KEEworlds, KEE and Knowledge Engineering Environment are trademarks of IntelliCorp.

[3]The same problem is obtained if equal numbers of intermediate subclasses are added along the two paths from Clyde to Elephants.

Inheritance mechanisms add derived results to a model. They also typically provide an efficient special-purpose form of truth maintenance for those results in that they remove information they have derived when a change occurs in the form or content of the hierarchies on which those derivations are based. If a general-purpose TMS has been incorporated into a frame system, then the TMS can be used to maintain the inherited information, thereby significantly reducing the complexity of the inheritance mechanism. However, such a reduction can be obtained only if the derivations performed during inheritance are expressible in the logical formalism supported by the TMS.

The inheritance algorithm in the current KEE system (and in other similar systems) is unsuitable for providing such justifications because it depends on arbitrary LISP procedures to perform its deductions and allows those procedures to use information whose semantic interpretation is unclear such as the order in which inheritance links are stored. The OPUS inheritance algorithm we present here performs sound deductions describable to a TMS in the form of nonmonotonic justifications whose justifiers are propositions expressible in the frame language. OPUS, therefore, in combination with the KEEworlds system, performs context-relative inheritance.

After presenting the formal description of the frame language, we present the OPUS inheritance algorithm in two forms. In the first form, the algorithm provides justifications to a TMS, which then manages the inherited information. In the second form, the algorithm performs its own truth maintenance and therefore is useable in a system that does not include a TMS.

II. A Frame Language with Defaults and Exceptions

The formal description we present for the OPUS frame language is based on the formalism developed by Etherington for default and exception links in inheritance networks [Etherington, 1987]. We found it desirable to extend Etherington's formalism in several ways to accommodate the structure of a frame language, to include more powerful constructs for describing exceptions to defaults, and to provide for the overriding of defaults at superclasses by defaults at subclasses. Those extensions are noted in our presentation and the motivations for them discussed.

A. The Language To Which Defaults Were Added

We begin by providing a set-theoretic description of the OPUS frame language before defaults and exceptions were added.

1. Frames

A *frame* represents an entity in the domain of discourse. Formally, a frame corresponds to a logical constant. A frame includes a collection of *own slots* that describe binary relationships considered to hold between the entity represented by the frame and other entities in the domain. A frame's collection of own slots necessarily includes MemberOf, which represents the standard set (i.e., class) membership predicate from set theory.

2. Class Frames

A *class frame* is a frame that represents a collection (i.e., class) of entities in the domain of discourse. Such a class is itself considered to be an entity in the domain of discourse. Thus, a class frame has associated with it a collection of own slots describing the binary relationships that the class has with other entities. Those own slots include Subclass, SubclassOf, Member, and MemberOf, which represent the standard subset and set membership predicates from set theory. These slots provide the "links" over which inheritance is done. In addition, a class frame has associated with it a collection of *prototype slots* that describe binary relationships considered to hold between each member of the class represented by the frame and other entities in the domain.

3. Own Slots

An own slot has associated with it a collection of *values*, each of which represents an entity in the domain of discourse. Formally, an own slot named S has associated with it a binary predicate, which for convenience we will also call S. An own slot S in a frame F having value V corresponds to the assertion *S(F,V)*.

4. Prototype Slots

A prototype slot has associated with it a collection of *necessary values*, each of which represents an entity in the domain of discourse. Formally, a prototype slot S has associated with it a binary predicate Nec$S$. A prototype slot S in a class frame C having necessary value V corresponds to the assertion Nec*S(C,V)*. Predicate Nec$S$ is related to predicate S by the following definition:[4]

$$\text{Nec}S(C, V) \equiv \forall x\, [\text{MemberOf}(x, C) \supset S(x, V)]$$

The following theorem follows from this definition and the set theory definition of SubclassOf in terms of MemberOf:

$$\text{Nec}S(C, V) \wedge \text{SubclassOf}(x, C) \supset \text{Nec}S(x, V)$$

That is, necessary values of a prototype slot at a class frame representing a class C are also necessary values of the prototype slot at all class frames representing subsets of C. The OPUS inheritance algorithm performs the deductions implied by the definition of Nec$S$ and by the theorem by propagating necessary values of prototype slots to all subclasses and class members.

The OPUS frame language without defaults can be characterized as expressing statements of the form *S(x,y)* and Nec*S(x,y)* for arbitrary first order binary predicates S. The language does not recurse in that it does not represent predicates of the form NecNec$S$.

B. Adding Defaults and Exceptions

Our goal was to augment the frame language described above to enable class frames to include prototypical descriptions of class members. That is, we wanted to enable prototype slots to have *default values* that would be inherited to class members as assumed values for the corresponding own slots unless blocked by *exceptions*.

We began by attempting to directly implement the formalism for defaults with exceptions in inheritance networks described by Etherington [Etherington, 1987]. Etherington's formalism is stated entirely in terms of unary class membership predicates. That is, he treats each class C as a unary predicate, *C(x)*, that is true when x is a member of C. He defines a "Membership" link between an object a and a class C to mean a belongs to class C (i.e., *C(a)*). The OPUS MemberOf own slot corresponds to the membership link. He defines a "Strict IS-A" link between class *C1* and class *C2* to mean *C1*'s are always *C2*'s (i.e., $\forall x\, [C1(x) \supset C2(x)]$). The OPUS SubclassOf own slot corresponds to the strict IS-A link.

Own slots are treated in Etherington's formalism by considering each slot-value pair *(S,V)* to be a unary predicate, *SV(x)*, corresponding to the class of all objects having value V for own slot S (e.g., the class of objects having color grey). Given that formalism for own slots, a necessary value V of a prototype slot S in a class frame C is a strict IS-A link between C and *SV*.

[4] Here and in the rest of the paper free variables are implicitly universally quantified.

Etherington represents default information in his inheritance networks by "Default IS-A" and "Exception" links. A default IS-A link from class *C1* to class *C2* means "Normally, *C1*'s are *C2*'s", and is expressed formally by the default logic inference rule:

$$\frac{C1(x) : C2(x)}{C2(x)}$$

The interpretation of this rule is: if *C1(x)* (called the *prerequisite*) is known, and *C2(x)* (called the *justification* where it appears above the line) is consistent with what is known, then *C2(x)* (called the *consequent* where it appears below the line) may be concluded.

An exception link has a class at its tail and a default IS-A link at its head. An exception link from class *C1* to a default IS-A link from *C2* to *C3* means "*C1*'s are exceptions to *C2*'s being *C3*'s" (e.g., "Royal elephants are exceptions to elephants being grey"). Etherington provides no independent semantics for an exception link. Instead, he defines it formally as a modification to the default rule corresponding to the link being blocked. However, Doyle has suggested (as reported by Touretzky [Touretzky, 1986]) that if the justification of the default rule corresponding to a default IS-A link contains an additional unary predicate unique to that default, then an exception link blocking that default can be defined to correspond to an assertion of the negation of that predicate for each member of the class at the tail of the link. Following that suggestion, a default IS-A link from class *C1* to class *C2* would correspond to the default rule:

$$\frac{C1(x) : C2(x) \wedge \neg \text{ExceptionTo}C1C2(x)}{C2(x)}$$

and an exception link from *C3* to the default IS-A link from *C1* to *C2* would correspond to the implication: $\forall x\,[C3(x) \supset \text{ExceptionTo}C1C2(x)]$.

To add Etherington's default IS-A and exception links to the OPUS frame language, we associate with each prototype slot in a class frame a set of *default values* and a set of *prototype exceptions*, and we associate with each own slot in a frame a set of *own exceptions*. Defaults consist of pairs of values and classes, where the class in the pair provides information to the inheritance mechanism indicating the class where the default originated. Prototype and own exceptions also consist of pairs of values and classes. An own exception *(V,OC)* for an own slot S in a frame F blocks inheritance to F of default value *(V,OC)* for S, and corresponds to the assertion "F is an exception to *OC* having S V" (e.g., "Clyde is an exception to elephants having color grey."). A prototype exception *(V,OC)* for a prototype slot S in a class frame C blocks inheritance to C and to all its members and subclasses of default value *(V,OC)* for S. Such a prototype exception corresponds to the assertion "Members of C are exceptions to *OC* having S V". For example, "Royal elephants are exceptions to elephants having color grey".

Formally, a default value *(V,C)* for a prototype slot S in the class frame C corresponds to the assertion Def*S(C,V)*, a default value *V,OC* for a prototype slot S in a class frame C representing a subclass of the class represented by class frame *OC* corresponds to the assertion SubDef*S(C,V,OC)*, an own exception *(V,OC)* for an own slot S in a frame F corresponds to the assertion OwnExc*S(F,V,OC)*, and a prototype exception *(V,OC)* for a prototype slot S in a class frame C corresponds to the assertion ProExc*S(C,V,OC)*. These predicates are related by the following definitions.

1. ProExc*S*

ProExc*S(C,V,OC)* means there is an own exception at each member x of C blocking the inheritance of default value V from class *OC* to own slot S in x. For a given binary predicate S, ProExc$S$ is defined as follows:

$$\text{ProExc}S(C, V, OC) \equiv$$
$$\forall x\,[\text{MemberOf}(x, C) \supset \text{OwnExc}S(x, V, OC)]$$

As was the case for predicate Nec$S$, the definition of ProExc$S$ implies that prototype exceptions are inherited to subclasses. That is:

$$\text{ProExc}S(C, V, OC) \wedge \text{SubclassOf}(x, C) \supset \text{ProExc}S(x, V, OC)$$

An assertion of the form ProExc*S(C,V,OC)* corresponds in Etherington's formalism to an exception link from C to a default IS-A link from *OC* to *SV*. OwnExc$S$ statements are inferred from ProExc$S$ statements and serve, following Doyle's suggestion, to block default rules at appropriate class members.

2. Def*S*

Def*S(C,V)* means that for each member x of C, if it is consistent to assume both that V is a value of own slot S in x and that no own exception at x blocks the inheritance of V for S from C, then it can be inferred that V is a value of own slot S in x. For a given binary predicate S, Def$S$ is defined as follows:

$$\text{Def}S(C, V) \equiv$$

$$\frac{\text{MemberOf}(x, C) : S(x,V) \wedge \neg \text{OwnExc}S(x, V, C)}{S(x, V)}$$

Def*S(C,V)* corresponds in Etherington's formalism to a default IS-A link from C to *SV*.

3. SubDef*S*

The SubDef$S$ predicate is an extension to Etherington's formalism to provide for the inheritance of defaults to prototype slots in subclasses. That is, the frame language is designed so that the prototype slots at any given class frame C have all the necessary and default values to be inherited by members of C that have been asserted at C or at any of C's superclasses. For example, the class frame AfricanElephants inherits from class frame Elephants the default value grey for the color prototype slot. Etherington has nothing in his formalism corresponding to that functionality.

For a given binary predicate S, SubDef$S$ statements are inferred from Def$S$ statements by the following axiom and default rule:

$$\text{Def}S(C, V) \supset \text{SubDef}S(C, V, C)$$

$$\frac{\text{SubDef}S(C,V,OC) \wedge \text{SubclassOf}(C,OC) : \neg \text{ProExc}S(C,V,OC)}{\text{SubDef}S(C,V,OC)}$$

Defaults asserted at a class as Def$S$ statements are used to infer SubDef$S$ statements at the class and are inherited to all subclasses as SubDef$S$ statements.

C. Quantified Exceptions

Etherington's link types and the statement forms we have introduced thus far for OPUS allow exceptions to be stated for specific values from specific origin classes. In practice, however, there is a need to assert collections of exception links. For example, one typically wants to state for a given slot in a given class frame (say the color slot in RoyalElephants) that *any* default value from *any* superclass is to be blocked and replaced by a given default value. Such assertions would be second order statements in Etherington's formalism. We can express them in the OPUS formalism as first order quantified statements as follows:

$\forall v$ OwnExc$S(O, v, OC)$
$\forall oc$ OwnExc$S(O, V, oc)$
$\forall v,oc$ OwnExc$S(O, v, oc)$

$\forall v$ ProExc$S(C, v, OC)$
$\forall oc$ [SubclassOf$(C, oc) \supset$ ProExc$S(C, V, oc)$]
$\forall v, oc$ [SubclassOf$(C, oc) \supset$ ProExc$S(C, v, oc)$]

The quantification of the origin class that is supported for prototype exceptions is only to *superclasses* of the class to whose members the exception applies. The restriction to superclasses is meant to implement the intuition that defaults at subclasses override defaults at superclasses. For example, a default color for royal elephants overrides a default color for elephants. Thus, we do not want a quantified prototype exception to block defaults from sibling classes and subclasses, but only from superclasses. (Although, note that the unquantified form of ProExc$S$ blocks defaults from any given class, including sibling classes and subclasses. The ability to block defaults from siblings may be useful in that it allows one to express a precedence ordering of defaults between classes even though their subclass-superclass relationship is unknown.)

As observed by Touretzky [Touretzky, 1984], the natural partial ordering of defaults in inheritance systems defined by the hierarchical structure of the inheritance graph resolves many ambiguities in an intuitive way. Touretzky introduces an "inferential distance" measure that expresses the desired natural ordering of defaults and uses that measure to filter out extensions that violate the ordering. In OPUS, that effect is obtained by the explicit quantification of exceptions over superclasses. In Touretzky's formalism, an exception always blocks a specific default value from all superclasses. Thus, unlike in OPUS, he cannot block all values from superclasses nor can he block values from a given superclass.

In summary, for any first order binary predicate S, the OPUS frame language represents statements of the following form (with their Etherington link equivalents where applicable):

$S(O, V)$ $\quad O>-\text{Member}->SV$
Nec$S(C, V)$ $\quad C>-\text{IS.A}->SV$
Def$S(C, V)$ $\quad C>-\text{Def.IS.A}->SV$
SubDef$S(C, V, OC)$
OwnExc$S(O, V, OC)$
$\forall v$ OwnExc$S(O, v, OC)$
$\forall oc$ OwnExc$S(O, V, oc)$
$\forall v, oc$ OwnExc$S(O, v, oc)$
ProExc$S(C,V,OC)$ $\quad C>-\text{Exc}->(OC>-\text{Def.IS.A}->SV)$
$\forall v$ ProExc$S(C, v, OC)$
$\forall v$ [SubclassOf$(C, oc) \supset$ ProExc$S(C, V, oc)$]
$\forall v, oc$ [SubclassOf$(C, oc) \supset$ ProExc$S(C, v, oc)$]

The system does not recurse in that it does not represent NecNec$S$, DefNec$S$, etc.

Consider how this formalism would be used to express the situation shown in Figure 1. DefColor statements would be used at Elephants and RoyalElephants to express the two defaults, and a quantified prototype exception statement would be used at RoyalElephants to block the inheritance of default colors from all superclasses, as follows:

DefColor(Elephants,Grey)
DefColor(RoyalElephants,White)
$\forall v, oc$[SubclassOf(RoyalElephants, oc)
$\supset$ ProExcColor(RoyalElephants, v, oc)]

III. A "Push" Inheritance Algorithm for Defaults and Exceptions

The OPUS frame language has been implemented by modifying the frame language in the KEE system. The inheritance mechanism implements the deductions defined by the definitions, axioms, and theorems given above by "pushing" necessary member slot values when they are asserted to subclasses and class members, and pushing default member slot values when they are asserted to subclasses and class members unless blocked by exceptions.

In this section we describe the algorithm in two forms, one assuming the availability of a TMS to maintain the derived results and the other not. In both cases we describe the information associated with each slot in the implementation and the operations performed by the algorithm.

A. What's In A Slot?

Each own slot in a frame has associated with it sets of values and own exceptions. Own exceptions are ordered pairs of the form (<value spec>, <origin class spec>), where <value spec> is either a value or the reserved symbol *, and <origin class spec> is either a class or the reserved symbol *. The * symbol matches any origin class or value and thereby corresponds to quantified own exceptions.

Each prototype slot in a class frame has associated with it sets of necessary values, default values, and prototype exceptions. Default values are ordered pairs of the form (<value>, <origin class>) and prototype exceptions are ordered pairs of the form (<value spec>, <origin class spec>). The * symbol in prototype exceptions matches any value or any origin class that is a superclass and thereby corresponds to the desired forms of quantified prototype exceptions.

B. Inheritance with a TMS

In order to perform inheritance using a TMS, each value or exception that is considered for a slot has an assertion (TMS node) associated with it. The assertion's formula (TMS datum) is as described in Section 2 for the different types of values and exceptions. A value or exception is added to a slot by giving its corresponding assertion a suitable justification, either a primitive justification or a justification recording some deduction external to the inheritance system. A given slot has a particular value or exception just in case the TMS assigns a positive belief status to its corresponding assertion. Demons are associated with each slot that are triggered by the TMS when an assertion concerning the slot is believed for the first time. A demon for a particular value or exception type is responsible for determining which inheritance justifications involving the newly believed assertion should be added to the TMS.

Necessary values of prototype slots are inherited to class members as values of own slots via justifications of the following form:

Nec$S(C, V) \wedge$ MemberOf$(Memb, C)$
$\rightarrow S(Memb, V)$

Necessary values of prototype slots are inherited to subclasses via justifications of the following form:

Nec$S(C, V) \wedge$ SubclassOf$(Csub, C)$
$\rightarrow$ Nec$S(Csub, V)$

Prototype exceptions are inherited from classes to class members via justifications of the following form:

ProExc$S(C, V, OC) \wedge$ MemberOf$(Memb, C)$
$\rightarrow$ OwnExc$S(Memb, V, OC)$

Prototype exceptions are inherited from classes to subclasses via justifications of the following form:

ProExc$S(C, V, OC) \wedge$ SubclassOf$(Csub, C)$
$\rightarrow$ ProExc$S(Csub, V, OC)$

Default values of prototype slots are inherited to class members as values of own slots via nonmonotonic justifications of the following form:

SubDef$S(C, V, OC) \wedge$ MemberOf$(Memb, C) \wedge$
OUT[OwnExc$S(Memb, V, OC)$]
$\rightarrow S(Memb, V)$

Note that there is no OUT justifier for $\sim S(Memb,V)$ in these justifications as the formal definition of default values requires. Such a justifier is not needed since statements of the form $\sim S(Memb,V)$ cannot be expressed in the frame language and are therefore necessarily out.

Default values of prototype slots are inherited to subclasses via nonmonotonic justifications of the following form:

SubDef$S(C, V, OC) \wedge$ SubclassOf$(Csub, C) \wedge$
OUT[ProExc$S(Csub, V, OC)$]
$\rightarrow$ SubDef$S(Csub, V, OC)$

As before, these justifications do not need to have an OUT justifier for $\sim$SubDef$S(Csub,V,OC)$ because statements of the form $\sim$SubDef$S(Csub,V,OC)$ cannot be expressed in the frame language and are therefore necessarily out.

Quantified own exceptions are used to generate instantiated own exceptions as needed to block the inheritance of default values that match the quantified form. The instantiated exceptions are produced via justifications of the following forms:

OwnExc$S(F, *, OC) \rightarrow$ OwnExc$S(F, V, OC)$
OwnExc$S(F, V, *) \rightarrow$ OwnExc$S(F, V, OC)$
OwnExc$S(F, *, *) \rightarrow$ OwnExc$S(F, V, OC)$

Quantified prototype exceptions are not inherited. Instead, they are used to generate instantiated prototype exceptions as needed to block the inheritance of default values that match the quantified form. The instantiated exceptions are produced via justifications of the following forms:

ProExc$S(C, *, OC) \wedge$ SubclassOf$(C, Csuper) \wedge$
SubDef$S(Csuper, V, OC) \rightarrow$ ProExc$S(C, V, OC)$
ProExc$S(C, V, *) \wedge$ SubclassOf$(C, Csuper) \wedge$
SubDef$S(Csuper, V, OC) \rightarrow$ ProExc$S(C, V, OC)$
ProExc$S(C, *, *) \wedge$ SubclassOf$(C, Csuper) \wedge$
SubDef$S(Csuper, V, OC) \rightarrow$ ProExc$S(C, V, OC)$

1. Example

Consider the statements that would be asserted and derived by this inheritance mechanism for the example from Figure 1. The inheritance of color Grey from Elephants to RoyalElephants would be done via the following justification:

SubDefColor(Elephants,Grey,Elephants) $\wedge$
SubclassOf(RoyalElephants,Elephants) $\wedge$
OUT[ProExcColor(RoyalElephants,Grey,Elephants)]
$\rightarrow$ SubDefColor(RoyalElephants,Grey,Elephants)

The inheritance of color Grey from Elephants to Clyde would be done via the following justification:

SubDefColor(Elephants,Grey,Elephants) $\wedge$
MemberOf(Clyde,Elephants) $\wedge$
OUT[OwnExcColor(Clyde,Grey,Elephants)]
$\rightarrow$ Color(Clyde,Grey)

The generation of the instantiated prototype exception for Grey at RoyalElephants would be done via the following justification:

ProExcColor(RoyalElephants,*,*) $\wedge$
SubclassOf(RoyalElephants,Elephants) $\wedge$
SubDefColor(Elephants,Grey,Elephants)
$\rightarrow$ ProExcColor(RoyalElephants,Grey,Elephants)

The instantiated prototype exception for Grey at RoyalElephants prevents inheritance of Grey as a default to RoyalElephants. Thus, no justification is generated for inheriting Grey from RoyalElephants to Clyde. Inheritance of the instantiated prototype exception for Grey at RoyalElephants to Clyde would be done via the following justification:

ProExcColor(RoyalElephants,Grey,Elephants) $\wedge$
MemberOf(Clyde,RoyalElephants)
$\rightarrow$ OwnExcColor(Clyde,Grey,Elephants)

That inherited exception would block the inheritance of Grey to Clyde.

The inheritance of color White from RoyalElephants to Clyde would be done via the following justification:

SubDefColor(RoyalElephants,White,RoyalElephants) $\wedge$
MemberOf(Clyde,RoyalElephants) $\wedge$
OUT[OwnExcColor(Clyde,White,RoyalElephants)]
$\rightarrow$ Color(Clyde,White)

Since there is no exception at Clyde blocking the inheritance of White from RoyalElephants, White will become the color of Clyde.

C. Inheritance Without a TMS

The above inheritance scheme relies on a TMS to remove inherited values when the assertions on which the inheritance was based are removed. For example, if the default color for elephants is removed, then the TMS will also remove Clyde's color if it was in the model only because of the default. Inheritance without the services of a TMS is considerably more complex since the inheritance machinery must, in effect, provide a truth maintenance capability for inherited values.

In order to provide for the removal of inherited values, the OPUS inheritance machinery requires each slot to have both a *local* and a *resultant* set of values and exceptions. The local sets are used only by the inheritance algorithm and contain those values or exceptions that are either asserted or are determined by some means other than inheritance. Resultant sets contain all the values and exceptions, including the local ones and those derived by inheritance. When a value or exception is to be added to (or removed from) a slot, it is added to (or removed from) the appropriate local set and the inheritance machinery recomputes the affected resultant sets for that slot. When the values of the MemberOf (or SubclassOf) own slot of a frame are modified, the inheritance machinery recomputes the resultant sets of each own slot (or prototype slot) of the frame. When a resultant set of a prototype slot is modified, affected resultant sets of all its descendants in the inheritance graph are recomputed. In the paragraphs below, we describe how each type of resultant set is computed. References in the descriptions to values and exceptions are to the resultant sets unless explicitly indicated otherwise.

The set of resultant necessary values for a prototype slot S in a class frame C is the union of the local set of necessary values for S in C and, for each *Csuper* that is a value of the own slot SubclassOf in C, the set of necessary values for prototype slot S in *Csuper*.

The set of resultant default values for a prototype slot S in a class frame C consists of the local default values for S in C and, for each *Csuper* that is a value of the own slot SubclassOf in C, the default values for prototype slot S in *Csuper* that do not match an exception for S in C.

The set of resultant values for an own slot S at a frame F consists of the local values for S at F and, for each C that is a value of the own slot MemberOf in F, the necessary values for prototype slot S in C and the default values for prototype slot S in C that do not match an own exception for S in F.

The set of resultant exceptions for an own slot S in a frame F is the union of the local set of exceptions for S in F and, for each C that is a value of the own slot MemberOf in F, the set of exceptions for prototype slot S in C.

The set of resultant exceptions for a prototype slot S in a class frame C consists of the local instantiated exceptions for S in C, for each *Csuper* that is a value of the own slot SubclassOf in C, the exceptions for prototype slot S in *Csuper*, and each *(V,Csuper2)* that matches a local quantified exception for S in C and is a default value for some *Csuper1* that is a value of the own slot SubclassOf in C.

Note that quantified exceptions remain in the local set and are not inherited. Quantified exceptions produce instantiated exceptions as needed to block defaults that would otherwise be inherited.

1. Example

Figure 2 shows the local and resultant values and exceptions produced by the inheritance algorithm for our elephants example. The default (Grey,Elephants) at Elephants and the quantified exception (*,*) at RoyalElephants would cause an instantiated exception (Grey,Elephants) to be generated at RoyalElephants. That instantiated exception would be inherited to Clyde. The exception at Clyde would block inheritance of the (Grey,Elephants) default from Elephants. The default (White,RoyalElephants) at RoyalElephants would be inherited to Clyde as Clyde's color.

IV. Conclusion

We have presented a formal description of a frame language that makes a clear distinction between necessary and default values of prototype slots. The formalization is based on previous work by Etherington, but extends his formalism to more closely match the structure of frame languages and to allow more convenient overriding of defaults at superclasses by defaults at subclasses.

We have presented two distinct methods for implementing the inferences warranted by the formal description of the frame language. The first makes use of nonmonotonic justifications in a TMS to record inferences corresponding to default inheritance. This method is suitable for situations in which a TMS is needed in order to maintain conclusions derived from non-inheritance inferences or to implement context-relative inheritance. The second method, in effect, implements a more efficient, special purpose truth maintenance algorithm in order to maintain the validity of inherited values. It is appropriate for situations in which a general purpose TMS is not needed.

A topic of current investigation is how to combine the two methods into a single system in which the special-purpose algorithm is used whenever possible. In many applications, general knowledge about the relationships among classes of objects in the domain and default values of prototype slots is entered directly by the domain expert and does not vary during the course of problem solving. The membership of individuals in classes and the values of own slots are more likely to be inferred during problem solving and to vary with hypothetical context. These considerations suggest that the special purpose algorithm can be used for maintaining inherited values in the upper regions of a taxonomy, with the TMS method being used as appropriate in the lower, more problem-dependent regions.

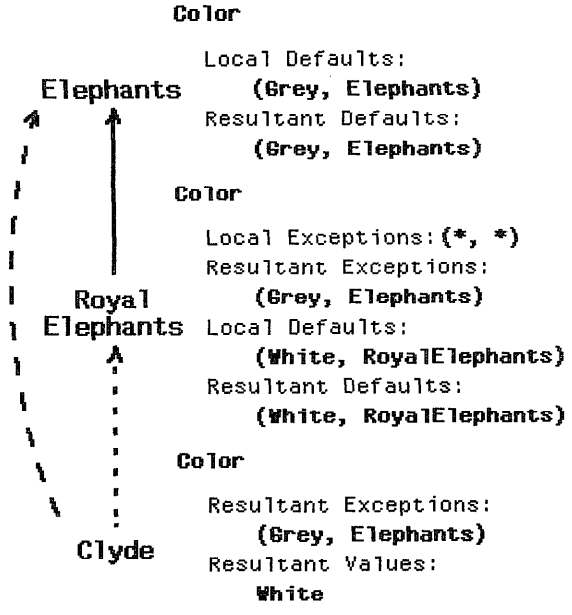

Figure 2: Inheritance without a TMS

References

[Brachman, 1985] Brachman, R.J.
"I Lied about the Trees" Or, Defaults and Definitions in Knowledge Representation.
AI Magazine 6(3):80-93, 1985.

[Brachman *et al.*, 1983] Brachman, R.J., Fikes, R.E., and H.J. Levesque.
KRYPTON: A Functional Approach to Knowledge Representation.
Computer 16(10):67-74, 1983.

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

[Doyle, 1979] Doyle, J.
A Truth Maintenance System.
Artificial Intelligence 12(3), 1979.

[Etherington, 1987] Etherington, D.W.
Formalizing Nonmonotonic Reasoning Systems.
Artificial Intelligence 31:41-85, 1987.

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

[Fikes and Kehler, 1985] Fikes, R. and T. Kehler.
The Role of Frame-Based Representation in Reasoning.
Communications of the ACM 28(9):904-920, 1985.

[Morris and Nado, 1986] Morris P.H., and R.A. Nado.
Representing Actions with an Assumption-Based Truth Maintenance System.
In *Proceedings AAAI-86.* Philadelphia, 1986.

[Roberts and Goldstein, 1977] Roberts, R.B. and I.P. Goldstein.
The FRL Manual.
MIT AI Memo 409, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, September, 1977.

[Touretzky, 1984] Touretzky, D.W.
Implicit Ordering of Defaults in Inheritance Systems.
In *Proceedings AAAI-84,* pages 322-325. Austin, Texas, 1984.

[Touretzky, 1986] Touretzky, D.W.
The Mathematics of Inheritance Systems.
Morgan Kaufmann, Los Altos, California, 1986.