FORMALIZING ATTRIBUTION BY DEFAULT

by
Paul C. Gilmore

Technical Report 87-26
July 1987

ABSTRACT: Attribution by default occurs when, in the absence of information to the contrary, an entity is assumed to have a property. The provision of information to the contrary results in the withdrawal of the attribution. It has been argued that classical logic cannot formalize such commonsense reasoning and that the development of a nonmonotonic logic is necessary. Evidence is offered in this note that this is not the case for some important defaults.

1. Introduction

To assert "All birds fly" is to attribute the ability to fly to each and every bird. From that assertion, and the knowledge that Tweetie is a bird, follows inevitably the attribution of the ability to fly to Tweetie. The discovery that Tweetie is a nonflying bird of some kind, say a penguin, results in a contradiction. But if the first attribution is intended as a default attribution, that is the intent of the assertion is "In the absence of evidence to the contrary, assume of each bird that it can fly", no contradiction results since the discovery that Tweetie is a penguin is evidence that Tweetie cannot fly.

It has been argued by Minsky in [1], by Reiter in [2], and by Etherington in [3], that this and other examples of attribution by default provide evidence for the failure of classical logic as a substitute for commonsense reasoning and that the development of a nonmonotonic logic is necessary. This short note demonstrates that this is not the case for some important defaults. A formalization of attribution by default is possible within a classical logic through the use of the standard database method of recording attribution, and the recognition of the subtleties of updating a database.

2. Purpose of Admitting Defaults

There is an underlying assumption in this paper as to the purpose of admitting attribution by default that should be explicitly stated.

A database management system is used by an enterprise to maintain a repository of data from which, through proper interpretation, the information essential to the operation of the enterprise can be obtained. The advantages cited by Date in [5] for the use of a database management system are the reduction of redundancy, assistance in the elimination of inconsistencies and inaccuracies, the sharing of data by many users, and the enforcement of standards and security restrictions. As the information recognized as essential to an enterprise grows in sophistication and subtlety, so too must the database. More advanced databases are now called knowledge bases, since the interpetation of the data recorded in them results in qualitatively different kinds of information from that in traditional databases. Nevertheless the advantages cited for database management systems must not be lost for knowledge base management systems intended for wide application.

Enterprises operate in a world in which much of what should be known is not, and in which some of what is known is incorrect. Database management systems must assist enterprises in reducing ignorance and

error. Such systems must be able to remind their users of missing data, and notify them of what might be unanticipated consequences of proposed updates. To the extent that it is possible for what may be a distributed repository, the internal consistency of the repository should be maintained. This paper is intended as a contribution to the development of such systems.

The development of knowledge bases for purposes other than for use by an enterprise may, of course, be necessary. For example, the development of a system having some of the characteristics of human knowlege bases, such as faulty reasoning, inconsistent beliefs, and leaps of imagination, may be essential for the testing of theories of human reasoning, but this paper is not intended as a contribution to such a development.

3. Predicates and Attributes

Predicate letters are used in first order logic to express membership in sets. For example, 'BIRD' may be used as a predicate letter of arity one to express in the assertion 'BIRD(x)' that what 'x' denotes is a member of the set of birds. A syntactic variant of this assertion is 'x:BIRD', where ':' has been used instead of the conventional 'ϵ' to express set membership. If 'Tweetie' is admitted as an individual constant, then 'Tweetie:BIRD' is true when'Tweetie' denotes the bird in question and 'BIRD' denotes the set of birds. In a similar fashion an assertion 'Tweetie:FLIES' can be used to express that Tweetie can fly. But the difficulties with attribution by default suggest that that is a poor formalization.

Using current terminology for database design, the set BIRD is an entity set, and FLIES is an attribute of BIRD. Depending upon the scope of the database to be developed, BIRD may be a primitive set without a larger set as its domain, or it may have a larger set say ANIMAL as its domain, meaning that BIRD is declared to be a subset of ANIMAL. In the second case FLIES may be an attribute of the larger set, and therefore also of BIRD, or it may be an attribute of BIRD but not of ANIMAL. Which of these options is chosen is irrelevent to the purposes of this note; it will be assumed that BIRD is a primitive set. More accurately, using the terminology of [6], BIRD is assumed to be a primitive <u>base</u> set. A base set, in contrast to a <u>defined</u> set, has an intension that can only be understood by humans. No machine can be programmed to recognize whether an entity is a bird or not; representatives of birds are added to BIRD at the request of users of the database. In contrast the number of members of BIRD can be determined at any time by the database system.

FLIES is an attribute of BIRD, not an entity set with domain BIRD,

although a subset of BIRD consisting of those birds that fly can be defined in terms of the FLIES attribute. An attribute is an association between an entity set and some set of strings called the value set for the attribute. For example, the value set VFLIES for the FLIES attribute will be defined to be:

(a) VFLIES for {v:STRING | v='F' ∨ v='NF' }.

Here STRING is a primitive defined set, or type, admitted for the database system, and '=' is identity for the type. Sequences of letters enclosed between single quotes are assumed to be names of members of STRING recognized by the system. The value set VFLIES, like all value sets, is a defined set. From the machine interpretable intension of VFLIES the system can determine the membership of VFLIES.

The FLIES attribute is a base set with domain BIRDxVFLIES; that is, the members of FLIES are ordered pairs ‹x,v› chosen by users, with x taken from BIRD and v from VFLIES. FLIES is base rather than defined because only a user can determine whether the entity represented by a member of BIRD is able to fly. However, not just any subset of BIRDxVFLIES is acceptable as the extension of FLIES; only one of the strings 'F' and 'NF' can be assigned to a member of BIRD. Therefore the following integrity condition must be satisfied:

(b) [∀‹x,u›,‹x,v›:FLIES] u=v.

Here '[∀‹x,u›,‹x,v›:FLIES]' is a generalized quantifier sanctioned in the language DEFINE introduced in [6]. The integrity condition would be expressed in conventional quantifier notation as follows:

[∀x][∀u,v](‹x,u›:FLIES ∧ ‹x,v›:FLIES ⊃ u=v).

The advantage of the generalized quantifier is that the narrowing of the range of the variables implicit in the antecedent of the conditional is made explicit.

It should be expected that a second integrity condition is also satisfied, since every bird either flies or does not fly:

(c) [∀x:BIRD][(∃v:VFLIES] ‹x,v›:FLIES.

But if a user is permitted to be silent on whether a particular bird such as Tweetie flies, then this condition clearly is not satisfied for FLIES. The possibility of databases in which (c) is not satisfied is the advantage obtained from the additional complication of treating FLIES as an attribute rather than as a subset of BIRD.

Although FLIES is a base set, a subset FLYING_BIRD of BIRD can be defined in terms of it:

FLYING_BIRD for { x:BIRD | ‹x,'F'›:FLIES }.

Just as the membership of VFLIES is determined by its machine interpretable intension, so also is the membership of FLYING_BIRD determined by its intension. But it is critical to note that a user cannot

update the membership of FLYING_BIRD directly, only the system can and does do that. But a user can update the membership of any base set, such as BIRD and FLIES, and so affect the membership of defined sets.

4. Attribution by Default

Assume that Tweetie has a representative which is a member of BIRD, and that 'Tweetie' is an individual constant which denotes that representative. To add the pair ‹Tweetie,'F'› to FLIES is to attribute the ability to fly to Tweetie. To assert

[∀x:B]‹x,'F'›:FLIES

is to attribute the ability to fly to every bird. If it is not true that every bird flies, then the assertion should not be made. If it is intended that a bird x should be assumed to be a member of FLYING_BIRD unless ‹x,'NF'›:FLIES is known to be true, then another definition of FLYING_BIRD is needed.

Consider the attribute DFLIES of BIRD defined as follows:

(d) DFLIES for {‹x,v›:BIRDxVFLIES | ‹x,v›:FLIES ∨
([∀u:VFLIES] ~‹x,u›:FLIES ∧ v='F') }.

Note that DFLIES satisfies (b) since FLIES does, and in contrast to FLIES, satisfies (c) as well, even when a user is silent on whether a bird flies or not. For example, if a user has added neither ‹Tweetie,'F'› nor ‹Tweetie,'NF'› to the membership of FLIES, then the assertions

‹Tweetie,'F'›:FLIES and ‹Tweetie,'NF'›:FLIES

are both false while the assertion

[∀u:VFLIES] ~‹Tweetie,u›:FLIES ∧ v='F'

is true when v is bound to 'F', since the assertion

~‹Tweetie,'F'›:FLIES ∧ ~‹Tweetie,'NF'›:FLIES ∧ 'F'='F'

is true. Therefore ‹Tweetie,'F'› is a member of DFLIES, even though it is not a member of FLIES. If DFLIES replaces FLIES in the definition of FLYING_BIRD,

(e) FLYING_BIRD for { x:BIRD | ‹x,'F'›:DFLIES }

then Tweetie is a member of FLYING_BIRD. The database system will respond to a query as to the membership of FLYING_BIRD with a list that includes Tweetie.

Should ‹Tweetie,'NF'› become a member of FLIES at some later time, either because a user explicitly adds it, or because it can be inferred from other assertions, then the definition of FLYING_BIRD assures that Tweetie is removed as a member. At that time the database system will respond to a query as to the membership of FLYING_BIRD with a list that does not include Tweetie.

In a similar fashion a don't know null value 'D/K' can be provided for

attributes whose values should be, but are not known. Let A be a base attribute on a set E with value set V. Assume also that 'D/K' is not a member of V and that V+ is V enlarged to include it. Let the set DA be defined:

DA for {‹x,v›Ex:V+ | ‹x,v›:A ∨ ([∀u:V] ~‹x,u›:A & v='D/K')}.

Then whenever a value for A has not been given for a member e of E, ‹e,'D/K'› is a member of DA. If a relation displaying the attributes of E is defined in terms of DA, rather than A, then 'D/K' would be recorded in the appropriate row and column as the value of DA.

The examples of default attribution described by Reiter in [2] and by Etherington in [3] can be dealt with in a similar fashion. Whether the full power of the default logic introduced by Reiter in [2], and the wider purposes of circumscription described by McCarthy in [4], can be realized remains to be seen.

5. A Formal Theory

A formal logic can be defined for those who favour this method of providing semantics for complex theories. The basis for the logic is the natural deduction based set theory NaDSet presented in [7], and the language DEFINE of the SET conceptual model described in [6] and [8] and motivated in [8] and [9]. The rules for first order quantification in NaDSet must be modified to accomodate the generalized quantifiers allowed in DEFINE.

In its second order form, the set INTEGER of integers can be defined within NaDSet. The members of INTEGER are available as representatives for entities; they be chosen to be the names satisfying the unique name axioms described by Reiter in [10]. In particular they may be chosen to represent the members of the set STRING. Alternatively, the first order form of NaDSet may be used and the unique name axioms added as additional true ground assertions. Ordered pairs can be defined in NaDSet, if desired, or for simplicitiy's sake may be assumed primitive.

By instructing the system to add members to base sets, users ensure that representatives of members are added to the sets. However, from the point of view of the system, the base sets, for any given state of the database, can be regarded as defined sets of representatives. For example, let $b_1, \ldots, b_k$ be the distinct representatives that have been chosen to be members of BIRD. Then the extension of BIRD, as far as the system is concerned, is determined by the definition:

BIRD for $\{u:INT \mid u=b_1 \vee \ldots \vee u=b_k\}$.

From this definition the ground atomic facts and the completion axiom for

the predicate BIRD described by Reiter in [10] are derivable in NaDSet using the set abstraction rules of the set theory.

A definition of VFLIES has already been given from which its ground atomic facts and completion axiom can be derived. The base set FLIES has its membership of pairs ‹b,v› determined by a definition similar to that of BIRD. Again the corresponding ground atomic facts and completion axiom are derivable. Further if the membership of FLIES has been chosen to satisfy the integrity constraint (b), then (b) is derivable in the theory.

Finally the memberships of the defined sets DFLIES and FLYING_BIRD are determined in NaDSet from their definitions (d) and (e) and their ground atomic facts and completion axioms are derivable.

Thus, if desired, a fully formal logic of attribution by default can be provided within a classical set theory. But because the base sets are restricted to being finite, the theory is unlikely to provide much help for computation, since quantification over such sets must be reduced to conjunctions and disjunctions of ground atomic facts. The theory does, however, provide some assurance that the ideas are sound, even though it is not a part of the implementation described by Morrison in [11].

Acknowledgment

Beneficial conversations with Roderick Morrison and George Tskinis are gratefully acknowledged as is the support of the Natural Science and Engineering Research Council of Canada.

Bibliography

1. Minsky, M., A framework for representing knowledge, P. Winslow (ed.), The Psychology of Computer Vision, McGraw-Hill New York (1975).
2. Reiter, R., A logic for default reasoning, Artif. Intell. 13 (1980) 81-132.
3. Etherington, D., Formalizing nonmonotonic reasoning systems, Artif. Intell. 31 (1987) 41-85.
4. McCarthy, J., Applications of circumscription to formalizing common-sense knowledge, Artif. Intell. 28 (1986) 89-116.
5. Date, C.J., An Introduction to Database Systems, Addison-Wesley, New York, (1981).
6. Gilmore, Paul C., The SET conceptual model and the domain graph method of table design, Dept. of Computer Science, University of B.C. Technical Report 87-7 (March 1987).
7. Gilmore, Paul C., Natural deduction based set theories: a new resolution of the old paradoxes, J. of Symbolic Logic 51 (1986) 393-411.
8. Gilmore, Paul C., Justifications and applications of the SET conceptual

model, Dept. of Computer Science, University of B.C. Technical Report 87-9 (April 1987).

9. Gilmore, Paul C., A foundation for the entity-relationship model: why and how? Dept. of Computer Science, University of B.C. Technical Report 87-11 (April 1987). To appear in the proceedings of the 6th Entity-Relationship conference (November 1987).
10. Reiter, Raymond, Towards a logical reconstruction of relational database theory, Michael L. Brodie, John Mylopoulos, Joachim W. Schmidt (eds) On Conceptual Modelling, Springer-Verlag Berlin (1984) 191-238.
11. Morrison, Roderick, Implementation considerations for a set based data model and its data definition/manipulation language. PhD thesis, Dept of Computer Science, University of British Columbia. In progress.