Default Reasoning and Inheritance Mechanisms on Type Hierarchies

Jaime G. Carbonell
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Type hierarchies abound in Artificial Intelligence, Data Bases and Programming Languages. Although their size, use and complexity differs, all share a central inference mechanism: **Inheritance of information**, their *raison d'etre*. This paper discusses various types of type hierarchies and inheritance mechanisms, concluding with a proposed *generalized inheritance mapping* approach to resolve issues of lateral and upward inheritance (to augment the traditional downward approach), as well as default reasoning and limited non-monotonic inference.

1. The Ubiquity of Type Hierarchies

Artificial Intelligence, Programming Languages and Data Bases share a common theme in their need and use of type hierarchies. For instance, artificial Intelligence combines type hierarchies with other declarative information and calls the result a semantic network. Programming languages and data bases tend to keep their hierarchies separate from other information. Type hierarchies always reflect both the nature of the data they classify and the operations performed upon it. The data-base world has by far the largest volume of data to classify; however, their data exhibits large-scale regularities and a high degree of homogeneity. At the opposite extreme, AI systems have encoded knowledge bases whose volume is orders of magnitude smaller than real-world data bases, but whose internal complexity, heterogeneity, and variety of operations performed upon the data are vastly greater.

This paper discusses various types of inheritance mechanisms in type hierarchies, followed by an attempt to unify these into a coherent, inheritance inference method. The discussion will be biased towards fulfilling present AI needs, but the concepts and mechanisms outlined here ought to be of general use, as programming languages and data bases increase the complexity and demands placed upon their type hierarchies. The discussion below is the summary of a synthesis derived from general AI folklore on Semantic Networks, including Quillian's original formulation [9], the SCHOLAR system [2], and more recently Hendrix [7], Fahlman [5], the KL-ONE system [1], Fox [6], and my own recent work on inheritance mappings (and also non-standard hierarchies [3]), and several enlightening discussions at the Data Abstraction Workshop.

2. Types of Type Hierarchies

There are various types of type hierarchies, depending upon whether strict inclusion is mandatory, whether a entry can have multiple types, whether type inclusion is necessarily transitive, exactly how the inclusion operation is defined, whether types partition a space (as opposed to arbitrary overlap), and whether default reasoning is permissible and/or monotonic in the inheritance mechanisms. Let me define some of the basic types of type hierarchies, which are closely coupled with the inheritance mechanisms discussed in the following section:

- *Single-Type inclusion with strict Partitioning (STP).* This is the simplest hierarchy, conceptually speaking. Every node has a single type. All types at each level are mutually-exclusive. The relation between a node and its type is the simple **member-of** relation. For example, most present-day type hierarchies in programing languages (without abstract or user-defined data types) fall into this category. If '4' is of type *interger*, it cannot also be of type floating-point. If *interger* is of type *number*, then by inheritance '4' is necessarily of type *number* and, by inheritance any legal operation defined for type *number* is a legal operation for type *interger* and for '4'.

- *Single-Type inclusion with No strict partitioning (STN).* This is essentially an STP, but there is no mutual exclusion enforced among the type categorizations. Data structure D can be of type *queue*, but that doesn't preclude it from being type *buffer* (same FILO properties) unless the builder of the hierarchy takes pains to enforce more precise definitions with the objective of producing an STP hierarchy.

- *Multiple-Type inclusion with No strict partitioning (MTN).* An MTN is like an STN except that each node can belong to more than one type. Therefore the induced hierarchy is a directed-acyclic graph (DAG) instead of a simple tree. For instance the node 'John' in an AI semantic network can belong to the types *male*, *US-citizen* and *Computer Programmer*. Each type, however is usually constrained to include only information that does not contradict entries in complementary type nodes.

- *MTN with default reasoning.* In any of the above hierarchies, including MTN, one can allow potentially contradictory information to be inherited from different types. The implications of relaxing the consistency constraint are discussed in the following section.

- *Arbitrary-Link Networks.* Heretofore, I assumed inheritance could proceed only from more abstract to more specific types, and there was a universal link type (i.e., ISA or SET-INCLUSION) through which all inheritance occurs. This need not be the case, as we see in the following section. Permitting generalized inheritance, however, yields a general graph (with cycles and potentially high connectivity) rather than a strict hierarchy.

3. Types of Inheritance Mechanisms

"Inheritance" means that assertions made of a type ought to be transmitted to all instances of that type. In general, inheritance is a transitive operation. Thus, if John is a man, men are land-animals, and land animals breath air, we can conclude, by inheritance, that John breathes air. However, this simple intuitive notion of inheritance is insufficient to represent a substantial fraction of all the information we would like to represent. For instance, consider a few classical problems, exemplified by the task of representing the following information in a type hierarchy:

1. **Birds fly, penguins are birds, penguins do not fly.**

2. **The climate of Panama is hot and humid, Colon is a city in Panama. Is it cold in Colon?** (How can we infer a negative answer?)

3. **Louis XIV tables have four ornate wooden legs and a heavy decorated wooden top. Each leg is attached with 8 wooden pegs. Tables consist of legs attached to a flat top. What are Louis XIV tables made out of? How many pegs do they have?** (The answers to both questions must be inferred.)

4. **A flivvit is just like a small car, except it has only three wheels arranged like a tricycle.**

5. **John is a graduate student. John is an heir to the Heinz fortune. Graduate students are not rich. Heirs to fortunes have a lot of money. Graduate students work hard. Heirs to fortunes do not work hard. Is John hard-working or rich or both or neither?**

The first case exemplifies the simplest form on non-monotonic inference in the inheritance process. Clearly, it is useful to store typical information with the type and note the few exceptions on the instances. Most birds fly, and if we did not know that penguins specifically do not fly, we would have concluded otherwise. Non-monotonicity means that the addition of new information can invalidate previously valid inferences. Here, the inheritance algorithm chooses the information stored lowest in the type hierarchy when conflicting information is found. (See Collins's LOK inference [4] and Reiter [10], for more discussion of non-monotonic default inference.)

The second and third examples suggest a need to inherit information along dimensions other than the traditional ISA or SET-INCLUSION found in practically all type hierarchies. More specifically, a part can inherit a homogeneous property of the whole (such as climate or language spoken in a country), but not other properties such as GNP - we cannot tell anything concrete about the economy of Boston soley from the U.S. GNP. In programming languages, for instance, it would be very useful to infer that an array (or any other homogeneous data structure) inherits properties from the type of it entries - if an element of an array is complex, the array is a complex array. Similarly, the inheritance mechanism ought to function from the whole to its parts.

The fourth example illustrates *lateral inheritance*, i.e., defining a type by means of modifying an existing type. Recall one of my types of type hierarchies definition "An MTN is like an STN except that..." The explicit representation of conceptual similarly (by stating only key differences) ought to augment the expressive ease and facilitate the extension of existing type hierarchies - especially in the data base and AI worlds, where large volumes of information may already be stored under existing type definitions. In general, lateral Inheritance can compose information from more more than one existing type (e.g., car and tricycle) to construct a new type (e.g., flivvit).

The last example illustrates a problem that has only recently arisen in AI. The entry for "John" wants to inherit information from more than one type entry - but this information may be in conflict. My proposed solution to this dilemma is to separate definitional from default inheritance - by explicitly stating which form of inheritance is applicable to which attributes in the inheritance mapping indexed by the type. Hence, "rich heir" states in its inheritance mapping that possessing a large amount of money is a definitional attribute, while supposed laziness is only a default attribute. The inference mechanism applies definitional inheritance first - which should never yield a contradiction - and default inheritance on a second pass only if it does not contradict[1] information concluded by the definitional inheritance pass. In cases where two different default inheritances contradict each other, I have a salience metric to determine which wins out - but I am on the market for a better resolution criterion but one that does not open the door to the full complexity of non-monotonic logic [8]

4. Concluding with General Inheritance Mappings

To summarize my brief discussion, I reiterate the key ideas and suggest where future research may lead. The simplest types of inheritance hierarchies afford the easiest type-checking and inheritance mechanisms - both from a conceptual-design criterion and efficiency of implementation. However, such hierarchies are much too limited for present day AI purposes and future developments in data bases and programming languages. The addition of default reasoning, tangled hierarchies, lateral inheritance and other mechanisms need not complicate the system excessively, especially if the bulk of the system's operations involve only simple definitional inheritance . My proposal, motivated in part by the recent work of Fox [6], is as follows:

- Inheritance can proceed *downward* - in the traditional fashion - *upward* - where the whole inherits from its parts, or a type is defined by enumerating its instances - or *lateraly* - where one type is defined as a modification or a composition of existing types. The directionality of the inheritance is an integral component of the inheritance mapping.

- Inheritance can be *default*, or *definitional*, or the attributes of a type can be divided into two sets according to which type of inheritance (if either) applies. This too is part of the inheritance mapping.

- Inheritance can proceed on different conceptual hierarchies, in addition to the traditional *ISA* hierarchy - for instance the *part-subpart* or *locational inclusion* hierarchies. Each dimension defines which properties are transferred by the inheritance mapping. (Even our old friend ISA does not inherit all properties: Kareem Jabbar, ISA man, man's average height is 5'10", but clearly Kareem Abdul Jabbar does not have an "average height" of 5'10".)

- Each type definition in the network has a pointer to the appropriate inheritance mappings - which essentially act as complex filters of information - to be used in inheritance operations through links to other concepts in the network.

- To make matters simpler (since I am an incorrigible believer in default reasoning), each and every component of the inheritance mapping may default to the preferences of the system's designer. My preferences are: inheritance proceeds *downward*, everything is *default*, and the traditional *ISA hierarchy* is chosen as the inheritance conduit - unless explicitly specified otherwise in an inheritance mapping.

In fact, the types of inheritance mappings can themselves be organized into a type hierarchy - with large savings in explicit storage of information. After all, that is the primary purpose of inheritance - to make common sense inference an automatic reflexive act without burdening the person (or mechanism) who must otherwise input large volumes of repetitious explicit knowledge. Furthermore, inheritance is very different from a general logic resolution theorem-proving method, in that it only makes one kind of inference - that which is most needed - and therefore does not suffer the unfavorable combinatorial properties of general logical inference.

Sophisticated inheritance mechanisms have proven to be useful tools in AI - in building real systems as well as in more theoretical work. While much work needs to be done, I sincerely hope that some useful interaction between AI and other disciplines starting to exploit more complex type hierarchies can develop. In particular, type inheritance mechanisms for rule-based systems, process descriptions, abstract data types, etc. appear to be indispensable as more human knowledge is encoded into these information structures.

5. References

1. Brachman, R. J., Bobrow, R. J., Cohen, P. R., Klovstad, J. W., Webber, B. L. and Woods, W. A., "Research in Natural Language Understanding," Tech. report 4274, Bolt Beranek and Newman, 1979.

2. Carbonell, J. R., "AI in CAI: An Artificial Intelligence Approach to Computer-Aided Instruction," *IEEE Trans. on Man-Machine Systems,* Vol. 11, 1970 , pp. 190-202.

3. Carbonell, J. G., "Towards a Process Model of Human Personality Traits," *Artificial Intelligence,* Vol. (in press), 1980 .

4. Collins, A., Warnock, E. H., Aiello, N. and Miller, M. L., "Reasoning from Incomplete Knowledge," in *Representation and Understanding,* Bobrow, D.G. and Collins, A., ed., New York: Academic Press Inc, 1975, pp. 383-415.

5. Fahlman, S. E., *NETL: A System for Representing and Using Real World Knowledge,* MIT Press, 1979.

6. Fox, M. S., "On Inheritance in Knowledge Representation," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence,* 1979 , pp. 282-284.

7. Hendrix, G., "Expanding The Utility of Semantic Networks Through Partitioning," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence,* 1975 .

8. McDermott, D. V. and Doyle J., "Non-Monotonic Logic I," *Artificial Intelligence,* Vol. 13, 1980 , pp. 41-72.

9. Quillian, M. R., "Semantic Memory," in *Semantic Information Processing,* Minsky, M., ed., MIT Press, 1968.

10. Reiter, R., "A Logic For Default Reasoning," *Artificial Intelligence,* Vol. 13, 1980 , pp. 81-132.

---

[1] Contradiction here is simply defined to mean different values for the same attribute. AI techniques for partial matching are being developed to determine degrees of contradiction/compatibility/corroboration, but these only serve to enhance the utility of default inheritance reasoning.