Inheritance and kinds of slots

A. HUTCHINSON
Department of Computing, King's College London, Strand, London WC2R 2LS

Kinds of slots, also called koses, were introduced previously by the author. A kos represents a binary relation. When relations are specifically portrayed thus in a data base, then novel inheritance mechanisms become feasible.

Received May 1987, revised October 1987

1. STANDPOINT ON FRAMES

For practical purposes, a data structure can be called a *frame* if it has the following features:

- It offers access to all information about one subject;
- The information is partitioned into values for *slots* within the frame;
- Each slot is of a named kind;
- A slot is uniquely defined by stating what kind of slot it is and which frame contains it;
- The frame as a whole can be treated as the set of its slots;
- The slot values may include frames, or ways of tracing them.

In all the literature about frames, a few key distinctions stand out. I shall take a particular position on three of them. Other stances are quite acceptable, but serve the present purpose less well.

Position 1. Frames, their representation and inheritance between them are worthy of study.

Not everyone agrees with this. Brachman suggests that inheritance is only an issue of implementation.[3]

Position 2. A frame will be treated as a passive data structure.

This contrasts with the stance chosen by Young and Proctor,[21] who treat a slot in a frame as an active entity which searches for its own value. They follow Lenat, who suggested that a slot may be regarded as a function on its frame. Our position will be modified slightly when we consider inheritance, which is an active process.

Position 3. A frame is a set of slots; and each slot holds a value and naught else.

Various authors have taken much more complex views. For instance, Winston and Horn give specific code for storing and retrieving many facets in each slot.[20] Charniak presents a relatively clear view, although his frames consist of more than just slots.[6] See Fikes and Kehler[9] for a survey. The slots considered here will have no facets. Most information which would reside in facets is kept in the slot's kind.

Knowledge representation by frames is rivalled by another technique, production systems, such as Prism.[17] The production system approach stemmed from a sound foundation, the notion of a grammar,[13] but has moved rather a long way from it since. So much good work has been done with it that it cannot be rejected out of hand; but productions are less attractive than frames because as Frost remarks, productions cannot describe a domain adequately.[10] They only describe actions upon data, and when to perform actions. Productions leave unanswered all questions about the form of data. (For this reason, Prism admits non-productions as data too.) This matters particularly for introspective programs, in which the program's data encompass the program's own actions.

In the author's previous article there is an outline of a database structure which includes a frame for each kind of slot, called a kos.[11] Slots themselves are usually regarded as featureless objects. This attitude is acceptable as long as one is not interested in programs which can extend their own type system. Unless the program is to have this capacity, all kinds of slots can be encoded rigidly. In fact, most authors do not distinguish a kind of slot from particular slots of that kind. For instance, Young and Proctor[21] write:

The slots *create*, *inst*, *read* and *write* have...special status.... The procedures which fill these slots are called 'interface procedures'.

Here, the second occurrence of the word *slots* has to refer to individual slots, because only an individual slot can be filled. The first occurrence refers to *kinds of slots* because *create* and *inst* are kinds, not particular slots. In this case there is no confusion because Young and Proctor regard a kind as no more than a name.

When a data base portrays koses thus, as 'first class' objects, any program using it has scope to create new kinds of relations between data. This is not true of current standard products. Koses also permit novel inheritance mechanisms, which will be described below.

2. INTERPRETATION

A frame base can be regarded as an *incomplete* portrayal of a theory. At any stage, the frame base depicts a finite subset of all theorems. This subset grows as new frames are created and new entries are added to slots. A *task* is an attempt to extend this subset.

Inserting an entry in a slot is like establishing a link in a network. Brachman, Fikes and Levesque describe two semantics for such links.[4] One treats a link as a *description*, which amounts to an axiom of the underlying theory. The other interprets a link as an *assertion*, which is either proved or else makes a statement about the 'real world'. There ought not to be any such dichotomy, since there is none in predicate calculus. I think their attitude has two causes:

(1) They distinguish between axioms, and theorems derived from the axioms. In classical predicate calculus, there is no such distinction.

Downloaded from https://academic.oup.com/comjnl/article/32/1/63/341956 by guest on 09 April 2024

(2) They distinguish between some abstract theory which may apply to many states of the real world, and a particular real world. Predicate calculus would normally treat both abstract and real world as parts of a single theory.

The approach presented here is meant to follow predicate calculus more closely. The theory encompasses both real world and abstract description. They may be distinguished by the user, if some concepts classify abstract theory and others classify real world entities, but the database portrays them all uniformly. Axioms correspond to the links in the initial state of the frame base; but when the program has added extra links, portraying derived theorems, the new links are indistinguishable from the old.

- Each atomic datum is a constant.
- A frame represents a compound term. Each creation calls some function with particular arguments. The created frame corresponds to the term whose leading symbol is this function, and with the arguments as subterms.
- A name of any frame is a constant. The program extend the theory's language by adding new names. Each name comes with an associated axiom: The name equals the term for its frame.
- A concept can be viewed in two ways. Firstly, it depicts a term, as does any frame. Secondly, if the concept can be defined by a predicate formula, then it also corresponds to the predicate which is its definition. (This predicate may be encoded in a language of functions, and stored in some slot in the concept's frame.) This dichotomy is acceptable. It is similar to the portrayal of predicate formulae by Gödel numbers, which are constants.

Among others, Charniak has viewed some (maybe all) frames as portraying one-place predicates.[6] In this treatment, those which do are concepts; those which do not are not. However, there is evidence that human concepts cannot always be defined thus.[12,18]

- A kos K whose internal format is *set* corresponds to a binary relation. If a slot $F.K$ of kind K occurs in frame F, then this relation is true of all pairs (F, x) where x is in the slot. It may be true of other pairs (F, y) as well. Any such y may be added to $F.K$.
- A kos whose internal format is *singleton* depicts a representable function, whose domain is the set of all possible frames with slots of this kind (including frames not yet created). The relation representing it would correspond to another kos, identical to the first except that its internal format is *set*.
- If K is a kos depicting a binary relation R, then the *dual kos* depicts the reverse of R.

Whenever a frame F_1 contains a slot of kind K, and the slot contains an entry which is a frame F_2, then F_2 may have another slot whose kind is DK, the dual of K. In this slot, there will be an entry F_1.

- The pattern of a concept contains all koses for relations and functions invoked in the concept's definition. The *pattern* kos is described in Hutchinson.[11]
- The *examples* kos depicts the satisfaction predicate S, which is defined in terms of a truth predicate T. This takes one argument $\langle P(t)\rangle$, where P is a formula with one free variable, and t is a term. S is true with arguments P and t if P is in positive form and is true on this term.

Thus entries in an *examples* slot represent theorems of the form

$$P(t).$$

Consistency of truth predicates is discussed by Kripke,[14] Martin[15] and Feferman.[8]

- The *generalisations* kos depicts implication between predicates. An entry in such a slot records a theorem of the form

$$\forall x . (P(x) \rightarrow Q(x))$$

where the slot is in the concept defined by P, and the entry is the name of the concept defined by Q.

- Storage is a way of recording theorems. If the user stores an entry in a slot, this entry plays the role of an axiom which extends the theory. If the program stores something at run time, then it is a derived theorem. Since the frame base only depicts a finite part of the theory, failure to retrieve is interpreted as 'unknown'.
- Creation will occur while the program tries to fulfil tasks. Each task can be phrased as

try to find an entity with certain properties P.

This corresponds to an attempt to prove some sentence of the form

$$\exists x . P(x).$$

The search for such an x is performed by creating terms which might satisfy P, until one does. Heuristics guide the choice of what to create. Universally quantified sentences cannot be proved directly by searching. If such a sentence $\forall y . S(y)$ is encoded, then the program can try to prove it by proving the equivalent:

$$\exists x . P(x)$$

where $P(x)$ means

'x is the name of a proof of $\forall y . S(y)$'.

There are two distinct sorts of predicate which may arise in this design. Predicates which define concepts must remain true, once established as true for a particular argument. The argument cannot become an example of the concept, and then cease to be one later on. These are the predicates of the underlying theory. The other sort of predicate can occur as a condition in a heuristic. Heuristics are designed to reduce entropy by selecting exceptional features. Something which appears exceptional at one stage of the program's evolution may seem normal later on. A condition in a heuristic can test the state of the frame base – e.g. has some concept got no *known* examples? It may not correspond to any feature of the underlying theory.

One can extend this formalism to cover relations of more than two arguments. Suppose that some such relation R is true on the tuple

$$(X_0, X_1, X_2, \ldots, X_n).$$

Then the program may contain an associated kos K, and a slot of its kind in the frame for X_0. The value of this slot is the set of such n-tuples $(X_1, X_2, \ldots, X_n)$. For each cyclic permutation of R, rotating left by i places, there will be another kos K_i whose slots will be found in frames X_i. This slot's value will contain n-tuples

$$(X_{i+1}, X_{i+2}, \ldots, X_n, X_0, \ldots, X_{i-1}).$$

The slot for a dual kos in K will contain the n-tuple of koses

$$(K_1, K_2, \ldots, K_n).$$

Downloaded from https://academic.oup.com/comjnl/article/32/1/63/341956 by guest on 09 April 2024

3. INHERITANCE

Each slot has two associated values: its *local* value, and its *full* value. The local value is what is stored directly in the slot by pointers in the computer's memory. The full value is the value employed in the frame base's interpretation, and includes all inherited contributions as well.

The classic form of inheritance is through a partial order, represented by IS-A or AKO links.[7, 20] This occurs between concepts. Brachman lists many of the confusions surrounding it.[3] Another form depends on INST links, which connect concepts with their instances. Some authors do not distinguish INST from IS-A.[19] As Frost said in his spoken presentation, even if all ambiguities and confusions surrounding IS-A were resolved, it would still not be adequate to describe all natural and practical forms of inheritance.[10] By contrast, the structure with koses allows much more versatile and realistic inheritance.

A common feature of inheritance algorithms is the following cycle. In order to find an inherited value for some slot $F.K$

First find some related frame F',
and a related kos K',
and then the local value of $F'.K'$ will contribute to the full value of $F.K$.

Many systems supplement this. KRL has a subtle searching pattern-matcher whose depth of search is chosen dynamically.[1] KEE permits simple deduction while searching (e.g. that if a slot has at least 16 values then it has at least 10 values. See Fikes and Kehler, p. 912).[9] Pure inheritance without any supplement is typified by Smalltalk.

In most published cases, it seems that

(*a*) $K' = K$ or else $K' =$ *default value*.
(*b*) F' is found from F through an IS-A or INST link, or something similar. In particular, the choice of F' is independent of K. There is only *one* path of entities F' from which F can inherit.

This is true for instance of Eiffel[16], and of Smalltalk even when supplemented with 'superclasses'.[2] The path may branch; for instance, if the IS-A slot in F holds several entities, then F may inherit from any or all of them. This is what Touretzky[19] and Cardelli[5] call 'multiple inheritance'. It will not be discussed here. Instead, we concentrate on how to find other frames F'.

Each kos K has a slot called its *inheritance path*. This holds another kos K'' (which may be the same as K). K'' is used to trace frames F' with contributing slots $F'.K'$. The kos K' may be either K or K'', depending on the particular inheritance strategy for K. The strategy will be a function, held in some other slot in K. There appear to be four standard strategies.

(1) *No inheritance*. The full value of $F.K$ is its local value, or a default. The value of K.*inheritance path* is *nil*.
(2) *Inheritance through itself*. In this case, K' and K'' are both K.

The frames F' are precisely those occurring in the full value of $F.K$. The internal format of K will be *set*, or some other format such as *list*, *bag*, *tree*, or *partial order* which has a natural projection onto *set*. Its entry type will be the type of F, or some generalisation of it.

Inheritance of a kos through itself appears to be fundamental. In all cases studied, K'' inherits through itself.

(3) *Inheritance through a parent kos*. In this case K' is K, and K'' is some other kos.

In F, there is another slot $F.K''$ which is inherited through itself. The frames F' are those occurring in the full value of $F.K''$. They all have the same type as F.

(4) *Inheritance through values*. In this case, K and K'' are distinct, and K' is K''.

As in (2), the frames F' are just those occurring in the full value of $F.K$. However, the entry type of K may not be the type of F, and there may be no slot of kind K' in F.

Case (1), no inheritance, is worth mentioning lest it be forgotten. Slots of kind *name* are never inherited.

The IS-A hierarchy provides an example of inheritance mechanism (2). The frame F is a concept, and K is *generalisations*. The generalisations of one concept are its immediate generalisations in the local value of $F.K$, and their immediate generalisations, and so on. The dual kos, *specialisations*, is also inherited through itself.

Another example of (2) occurs among heuristics. A heuristic is said to be 'weak' if it can be applied in many cases, and 'strong' if its conditions are restrictive. In each heuristic's frame H, there is a slot called *weaker heuristics* whose full value is the set of all other heuristics with weaker conditions. This slot is inherited through itself. So also is its dual, *stronger heuristics*.

The *examples* slot in any concept C is inherited through its parent kos, *specialisations*. The full value of C.*examples* consists of its local value together with the local examples in all specialisations of C.

Each heuristic H has a slot of *conditions*, which is also inherited through a parent, *weaker heuristics*. Its value is a partial order with each node a particular condition. H is only applicable if all conditions are fulfilled. The conditions are ordered because some of them only make sense when others hold.

The dual kos of *examples* is *inst*. This is inherited through its values. The local value of F.*inst* is a set of concepts C which have F as an example. The full value of F.*inst* consists of these concepts and all their generalisations, so K' and K'' for *inst* are both *generalisations*.

Each condition for heuristics has a frame of its own, say Cond. This contains a slot called *restricted heuristics*, whose full value contains all heuristics whose scopes are restricted by Cond. This slot is also inherited by mechanism (4), through its own values. The local value of Cond.*restricted heuristics* contains a few heuristics H. The full value contains these heuristics H, and all stronger ones too. The inheritance path of *restricted heuristics* is *stronger heuristics*.

Among these examples, a pattern is emerging.

– If K is inherited through itself, then so is its dual kos.
– If K is inherited through a parent kos, then the dual of K is inherited through its own values, and vice versa. Moreover, the inheritance paths of K and its dual kos are themselves duals.

Inheritance mechanisms can be described in terms of the relations to which koses correspond. Say kos K

Downloaded from https://academic.oup.com/comjnl/article/32/1/63/341956 by guest on 09 April 2024

corresponds to relation R. An object y will be in the full value of slot $F.K$ iff FRy is true. Say we define another relation Fry, which is true iff y is in the local value of $F.K$. Also, say P is the transitive relation corresponding to the inheritance path K'' of K.

(1) No inheritance:
$R = r$.
(2) Inheritance of K through itself:
R is the transitive closure of r.
(3) Inheritance of K through a parent K'':
R is the smallest relation for which
(*a*) $\forall x \forall y\, (xry \rightarrow xRy)$
and
(*b*) $\forall x \forall y \forall z\, (xPz \,\&\, zry \rightarrow xRy)$.
(4) Inheritance through values:
R is the smallest relation for which
(*a*) $\forall x \forall y\, (xry \rightarrow xRy)$
and
(*b*) $\forall x \forall y \forall z\, (xrz \,\&\, zPy \rightarrow xRy)$.

Proposition 1

In (3) and (4), the conditions (*b*) still hold when r is replaced by R:

($3b'$) $\forall x \forall y \forall z\, (xPz \,\&\, zRy \rightarrow xRy)$
($4b'$) $\forall x \forall y \forall z\, (xRz \,\&\, zPy \rightarrow xRy)$.

Proof

In both cases, it follows from the fact that P is transitive.

It is interesting to conjecture what other forms of inheritance might occur. They would differ in their closure condition (*b*). If the inheritance path can only consist of one other kos, then (3) and (4) seem to be the only possibilities. If we allow two koses in the inheritance path, corresponding to relations P and Q, then the closure condition could be any of

($5b$) $\forall x \forall y \forall z_1 \forall z_2\, (xrz_1 \,\&\, z_1 P z_2 \,\&\, z_2 Q y \rightarrow xRy)$
($6b$) $\forall x \forall y \forall z_1 \forall z_2\, (xPz_1 \,\&\, z_1 r z_2 \,\&\, z_2 Q y \rightarrow xRy)$
($7b$) $\forall x \forall y \forall z_1 \forall z_2\, (xPz_1 \,\&\, z_1 Q z_2 \,\&\, z_2 r y \rightarrow xRy)$.

These new mechanisms are distinct from the first four. Only (6) satisfies an analogous proposition, because the relation between x and y defined as

$$\exists z\, (xPz \,\&\, zQy)$$

may not be transitive, even when P and Q are. Any more elaborate condition containing three or more parents will not satisfy it either.

Proposition 2

If r is given, and $P_1, \ldots, P_n$ are arbitrary transitive relations, and R is the smallest relation for which

(*a*) $\forall x \forall y\, (xry \rightarrow xRy)$

and

(*b*) $\forall x \forall y \forall z_1 \ldots z_n$
$(xP_1 z_1 \,\&\, \ldots z_{i-1} P_i z_i \,\&\, z_i r z_{i+1} \,\&\, \ldots z_n P_n y \rightarrow zRy)$

then the only cases in which R also satisfies

(b') $\forall x \forall y \forall z_1 \ldots z_n$
$(xP_1 z_1 \,\&\, \ldots z_{i-1} P_i z_i \,\&\, z_i R z_{i+1} \,\&\, \ldots z_n P_n y \rightarrow xRy)$

are (3), (4) and (6) above.

I have not discovered a practical case of mechanism (6).

4. ELABORATIONS ON INHERITANCE

Inheritance is relatively easy and natural through a tree. Some inheritance paths form partial orders which are not trees. Then inheritance is subtler, but still possible.[16, 19]

There is a further complication. Suppose the program invents concepts by composing their defining predicates. At some stage, it may discover a proof that two or more such predicates are equivalent. If nothing prevents it, then it will create a loop in the network of *specialisations*, which would put all inheritance strategies through *specialisations* into a loop.

The obvious solution is to allow each concept to have several alternative definitions. If ever such a loop of equivalent predicates is discovered, then rather than link their concepts in a loop, the program could amalgamate their defined concepts.

Another solution would be to distinguish some one definition and associated concept as *principal*, and let the other concepts remain in the frame base as *alternatives*. An alternative could have examples but could not have local specialisations of its own. All its specialisations would be traced through its equivalent principal.

There may well be a natural hierarchy of koses, or maybe more than one. Most slots in a kos could be inherited; but *inheritance path* is an exception. Certainly the inheritance path of the kos *inheritance path* cannot be inherited.

Inheritance of heuristics' conditions presents another subtlety. Recall, conditions are partially ordered by applicability. One condition may only be calculable when all weaker conditions are known to hold. Each condition has its own formal parameters, which are bound to actual values when the condition is evaluated. It is essential that the same value is bound to corresponding parameters in comparable conditions.

The relation *is weaker than* should not just relate two conditions. It should also relate each formal parameter in the weaker condition to a formal parameter of the stronger.

Acknowledgements

This paper has benefitted from pleasant discussions and correspondence with Richard Bartle, Malcolm Bird, Tony Brooker, Simon Croft, Jim Doran, Solomon Feferman, Donald Perlis, Richard Spencer-Smith and David Till.

REFERENCES

1. D. G. Bobrow and T. Winograd, An overview of KRL, a knowledge representation language. *Cognitive Science* **1**, 3–46 (1977).
2. A. H. Borning and D. H. H. Ingalls, Multiple inheritance in Smalltalk-80. *AAAI-82 Proceedings,* Pittsburgh, pp. 234–237 (1982).

Downloaded from https://academic.oup.com/comjnl/article/32/1/63/341956 by guest on 09 April 2024

3. R. J. Brachman, What IS-A is and isn't. *Computer* **16** (10), 30–36 (Oct. 1983).
4. R. J. Brachman, R. E. Fikes and H. J. Levesque, Krypton: A functional approach to knowledge representation. *Computer* **16** (10), 67–73 (Oct. 1983).
5. L. Cardelli, A Semantics of Multiple Inheritance. In *Semantics of Data Types*, edited G. Kahn, D. B. MacQueen and G. Plotkin, Lecture Notes in Computer Science no. 173, pp. 51–67. Springer, Heidelberg (1984).
6. E. Charniak, A common representation for problem-solving and language-comprehension information. *Artificial Intelligence* **16**, 225–255 (1981).
7. E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA (1985).
8. S. Feferman, Toward useful type-free theories. Part 1. *The Journal of Symbolic Logic* **49** (1), 75–111 (March 1984). (See also Ref. 15.)
9. R. Fikes and T. Kehler, The Role of Frame-Based Representation in Reasoning. *Communications of the ACM* **28** (9), 904–920 (1985).
10. David P. Frost, A natural language interface for expert systems: system architecture. In *Advances in Artificial Intelligence: Proceedings, AISB*, Edinburgh, pp. 157–168. Wiley, Chichester (April 1987).
11. A. Hutchinson, A data structure and algorithm for a self augmenting heuristic program. *The Computer Journal* **29** (2), 135–150 (April 1986).
12. P. N. Johnson-Laird, *Mental Models*. Cambridge University Press (1983).
13. R. Kurki-Suonio, *Computability and Formal Languages*. Auerbach (1971).
14. S. Kripke, Outline of a theory of truth. *The Journal of Philosophy* **72**, 690–716 (1975) (See also Ref. 15.)
15. R. L. Martin, Editor's Introduction to *Recent Essays on Truth and the Liar Paradox*, pp. 1–8. Oxford University Press (1984).
16. B. Meyer, *Object-oriented software construction*. Prentice Hall (1988).
17. S. Ohlsson and P. Langley, *Prism Tutorial and Manual*. University of California, Irvine (1986).
18. S. L. Peters and S. C. Shapiro, A representation for natural category systems. *Proceedings IJCAI 87*, Milan, vol. 1, pp. 140–146 (August 1987).
19. D. S. Touretzky, *The Mathematics of Inheritance Systems*. Pitman (1986).
20. P. H. Winston and B. K. P. Horn, *Lisp*. Addison-Wesley, Reading, MA (1981, 1984).
21. S. J. Young and C. Proctor, UFL: An experimental frame language based on abstract data types. *The Computer Journal* **29** (4), 340–347 (1986).

Downloaded from https://academic.oup.com/comjnl/article/32/1/63/341956 by guest on 09 April 2024