Using Semantic Concepts to Characterise Various Knowledge Representation Formalisms: A Method of Facilitating the Interface of Knowledge Base System Components

R. A. FROST
Department of Computer Science, University of Glasgow

Currently, there are a number of research groups working on various components for knowledge base system (KBSs). As example: (a) novel hardware is being developed for mass storage of simple facts, (b) machines are being built to speed up reasoning with rules expressed in languages such as PROLOG and LISP, (c) algorithms have been designed for automatic maintenance of semantic integrity and for deductive question answering, (d) logical systems are being axiomatised which can accommodate time, beliefs, non-monotonic reasoning and other aspects of knowledge which cannot be handled by classical truth-functional predicate logic, (e) methods are being developed to support multiple user-views of knowledge stored in some canonical form, and (f) some progress has been made in providing natural-language interfaces to knowledge base systems.

Integration of such components is problematical for a number of reasons, not the least of which is due to the different terminologies and knowledge representation formalisms which are used by the various components. A possible solution to this problem is to identify a commonly used set of semantic concepts and then employ this set of concepts to characterise the type of knowledge which is processed by the various components. An example of a semantic concept is logical negation (i.e. not). Some knowledge representations, such as those used in classical logic, can accommodate logical negation whereas those used in conventional database systems are unable to represent logical negation other than by omission in conjunction with the closed-world assumption.

Choice of an appropriate set of semantic concepts should be based on pragmatic criteria rather than philosophical argument, otherwise it is unlikely that agreement will be reached on what concepts to include. In this short paper we present a version (0) set of concepts which was chosen intuitively. We illustrate how this set might be refined by application to example components of KBSs.

This paper is a revised version of a paper presented at the 2nd Alvey-sponsored Workshop on Architectures for Large Knowledge Bases (WALKB2) held at Manchester University and organised by Simon Lavington. At that workshop it was agreed to pursue the approach outlined in this paper by setting up a study group consisting of representatives from industry and academic institutions. The remit of this group is to refine the set of semantic concepts by application to a range of knowledge representations including those used in database models, various formal logics, semantic nets, production systems, logic programming languages, hardware-based systems and so on. The initial output from this group will be the version (1) set of well-defined semantic concepts which all knowledge base research groups will be encouraged to use to characterise the particular components which they are developing. The version (1) set of semantic concepts is scheduled to be available mid 1985.

1. KNOWLEDGE BASE SYSTEMS

A *knowledge base* is a collection of simple facts such as 'John works for IBM' together with general rules such as 'all humans are either male or female'. A *knowledge base system* (KBS) is a set of resources: hardware, software, and possibly human, whose collective responsibilities include storing the knowledge base, maintaining security and integrity, and providing users with the required input/output routines, including deductive retrieval facilities, so that the knowledge base can be accessed as required. Knowledge base systems, as currently discussed in the literature, are distinct from *conventional* database systems in four ways:

(*a*) knowledge bases contain *explicitly represented* rules as well as simple facts;

(*b*) knowledge base storage structures have low *structural semantic content* compared with database structures;

(*c*) knowledge base systems include components for the *automatic maintenance of semantic integrity* in addition to components for syntactic checking as found in conventional database systems;

(*d*) knowledge base systems include components which can make inferences over the knowledge base, thereby providing a *deductive retrieval* facility.

KBSs are also distinct from *expert systems* which are typically designed for specific tasks such as mineral prospecting, medical diagnosis, fault-finding and mathematical theorem proving. KBSs might be used as components in expert systems. However, their use is not limited to this. They can be used as general-purpose sophisticated database systems or as components of 'special function' systems such as pattern-recognition systems.

The distinction between KBSs and *fifth-generation database systems*, as defined by Nijssen is not so clear.[1] A reasonable solution is to regard fifth-generation database systems as belonging to a particular type of KBS in which the *rules are relatively few and relatively static*. The notion of conceptual *schemas* in fifth-generation database systems reflects the stability of the general rules and the use to which the rules are put.

2. CURRENT DEVELOPMENTS IN KBS COMPONENTS

Research groups are currently working on various aspects of KBSs. These include the following.

(*a*) Hardware is being developed for the mass storage of simple facts represented in data structures with low structural semantic content.[2, 3, 4]

Downloaded from https://academic.oup.com/comjnl/article/28/2/112/343415 by guest on 10 April 2024

(*b*) Hardware is being develped to speed up reasoning with rules expressed in languages such as Prolog and LISP,

(*c*) Methods are being developed for the automatic maintenance of the semantic integrity of knowledge bases using rules expressed in languages based on first-order predicate logic.[5,6]

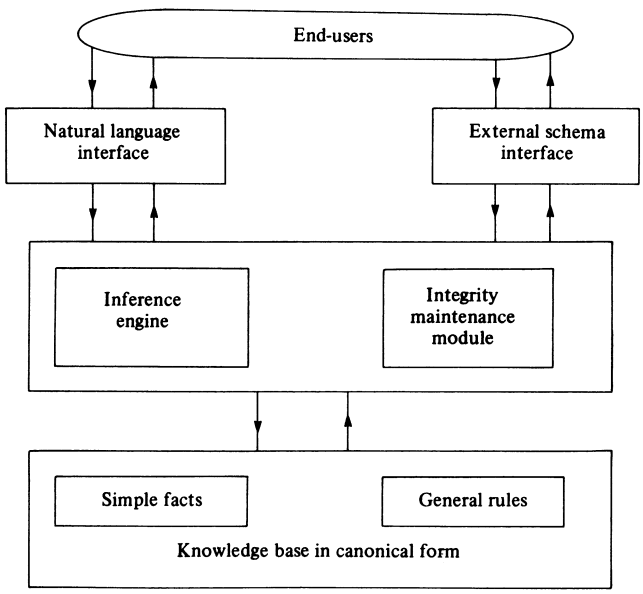
(*d*) Methods have been developed to speed up deductive retrieval by mixing theorem-proving techniques from sorted first-order predicate logic with relational algebraic operations such as division and projection as used in relational database systems.[7,8] Extensions to Prolog are also being investigated. For example, a sorted logic has been developed by Cohn and is being implemented as an extension to Prolog,[9] and a version of Prolog which incorporates the notion of a schema (called a model in the paper) has been developed by Babb.[10]

(*e*) The use of logic to express and reason with knowledge involving uncertainty, beliefs, time, and so on, is being investigated.[11,12,13]

(*f*) Methods have been develped which allow multiple user-views (external schemas) of knowledge which is stored in some standard canonical form.[14,15] Also some progress has been made in the automatic translation of statements expressed in natural language to statements expressed in a canonical form.[16,17,18]

3. INTEGRATION OF COMPONENTS

Integration of components such as those mentioned above could result in a KBS with the structure shown in the diagram below. Such an architecture has much in common with that proposed by Nijssen for fifth-generation DBMSs, which is itself an extension of the architecture published in the report of the International Standards Organisation.[1,19]



4. A PROBLEM

The difficulty in constructing such a system is in interfacing (and/or integrating) the components and techniques which have been developed by the various research groups. Such interfacing will be problematical for a number of reasons. For example: incompatible hardware, operating systems and programming languages will create difficulties. However, the solution to such problems is relatively straightforward. A greater obstacle will result from the different knowledge representation formalisms which are used by the various components. For example, consider interfacing a sorted first-order logic theorem prover (such as that described in Cohn)[9] with the IFS data store (as described by Lavington).[2] It is not at all obvious that such an interface is possible or even desirable. The compatibility or incompatibility of these components is obscured by the different terminology which is used to describe them and by the different notations which they use for the representation of knowledge.

5. A POSSIBLE SOLUTION

A possible solution to the problem is to identify a commonly used set of *semantic concepts* which can be used to characterise the type of knowledge which can be represented and processed by particular KBS components. Note that we are *not* suggesting that we should try to identify a best approach for the representation of knowledge *nor* are we suggesting that we should try to identify a universally acceptable set of semantic primitives. We are simply saying that we should try to identify a set of comonly used semantic concepts which can be used to characterise various knowledge representation formalisms.

What do we mean by semantic concepts?

The term *semantic* has to do with meaning rather than form. The word *concept* is defined as 'an abstract notion; a mental impression of an object'. A semantic concept can, therefore, be thought of as an abstract notion which can be represented in various ways using syntactically different notations. An example of a semantic concept which is relevant to the present discussion is *logical negation*, which can be represented in various ways. For example:

(*a*) not;
(*b*) ¬;
(*c*) it is false that;
(*d*) by omission.

The last of these representations involves the *closed-world assumption* as described by Reiter.[7]

Other semantic concepts which *might* be of relevance include the following:

propositions (semantically indivisible statements which are true or false)
truth
falsity
unknown truth-values
uncertainty values
entities (sometimes called objects)
entity-sets
attributes
attribute-sets
relations
relationships (n-tuples, $n \geqslant 1$)
functions
names

Downloaded from https://academic.oup.com/comjnl/article/28/2/112/343415 by guest on 10 April 2024

variables (free variables)
universally quantified variables
existentially quantified variables
not (logical negation)
and
or
material implication (→, if...then)
possible worlds, possible truth, necessary truth, strict implication } from modal logic
agent (the 'knower' or 'believer' in belief logic)
proof (as used in non-monotonic logic)
entity-set membership rules
relation domain cardinality rules
rules which can be expressed as Horn formulas

and so on.

In order to avoid ambiguity, it would be useful to name and define semantic concepts using some well-established terminology such as that found in Marciszewski's *Dictionary of Logic* (1981).[20]

The concepts listed above were chosen intuitively. It is recognised that they will require a good deal of re-working before a really useful set can be identified. The remaining part of this short paper illustrates one method by which the set of concepts might be refined and extended. The method simply involves using the set on some typical KBS components and modifying it accordingly. We feel that this pragmatic approach is likely to be more rewarding than lengthy philosophical discourse on what does or does not constitute a semantic concept.

6. THE HIERARCHICAL APPROACH

In the basic hierarchical approach, knowledge is represented by the use of files, records and fields related in tree structures. An example of a hierarchical structure is given in the diagram below.

Note. An instance of a record is only allowed to occur in one tree and is only allowed to have one 'parent' in that tree. An instance of a field (e.g. welding) may occur in more than one tree and may have more than one parent in any tree.

Using some of the semantic concepts mentioned earlier, we can characterise the hierarchical approach as follows.

Semantic concept	*Corresponding hierarchical terminology/concept*
entity	record instance
attribute	field value
entity-set	record type
attribute-set	field type
one-to-many relation between entity sets	record links between father and son records in files
many-to-many relation from entity set to attribute set	record-field links
negation	(by omission)
static entity-set membership rules	record type structure
static attribute-set membership rules	field format
relation domain and counter-domain rules	field specification, record specification

The representation of rules in the hierarchical approach is embedded in the structure of the files, records and fields:

(*a*) entity-set membership rules are implicitly defined by restricting records to have a particular structure;

(*b*) attribute-set membership rules are implicitly defined by restricting fields to have a fixed format;

(*c*) relation domain and counter-domain rules are implicitly defined by restricting the type of records allowed in files and by restricting the types of field allowed in records.

Negation may only be represented by omission if it is appropriate to make the closed-world assumption. From this example, we can see that our list of concepts should include specific types of rule such as entity-set membership rules.

7. PROLOG

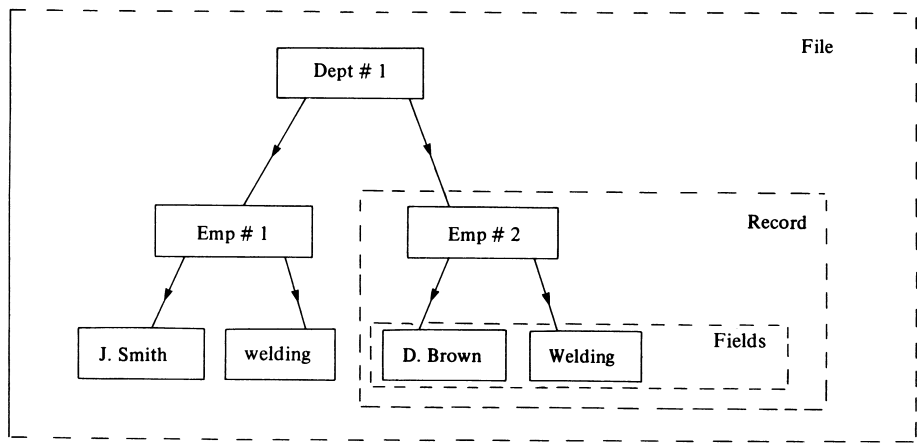
PROLOG programs are sets of Horn clauses of first-order predicate logic.[21] To run a program, the calculation to be made (or question to be answered) is made a goal clause and proved by a lush resolution theorem prover which uses a depth-first search strategy. Some special predicates such as the 'write' predicate do not use resolution but are evaluated by separate routines and then deleted.

Using some of the primitives above, PROLOG may be characterised as follows:

semantic primitives underlying PROLOG

entities
unrestricted n-ary relations ($n \geqslant 1$) + functions
variables, and, or, material implication } use restricted to → ⌐ ↓
any rule which can be expressed as a Horn clause
negation by omission (failure to prove)

Note. No distinction is made between entities and attributes. There is no notion of an entity-set in



Downloaded from https://academic.oup.com/comjnl/article/28/2/112/343415 by guest on 10 April 2024

'standard' PROLOG. Any rule which can be expressed as a Horn clause can be represented in PROLOG. The 'not' operator in PROLOG really means fail to prove. There is no way of expressing, for example, the fact that 'John is not married to Sally' other than by omission of the fact that John is married to Sally or by introducing a predicate 'notmarriedto'.
This example to also demonstrates the need for our list of concepts to include more specific types of rule.

8. THE FACT MACHINE

The FACT machine is a knowledge storage structure which is being developed in VLSI.[3] The knowledge is represented as labelled binary-relationships where the label may be used to represent quantification or to represent the context in which the relationships are relevant. For example:

F#1	John.is a		. fireman
F#2	Bill.thinks		. F#1
F#3	bridges.carry		. traffic
F#4	F#1.start time		. 3/4/83
F#5	The Forth Road Bridge	. ε	. bridges
F#6	carry	. ε	. transport

The binary-relationship labelled F#3 is a 'generic' rule which may be read as 'all bridges carry traffic'.
The FACT machine can be characterised by the following concepts:

concepts underlying the FACT machine
entities
binary-relations between entities *and* members of relations (i.e. binary-relationships)
'generic' rules such as that given above
negation by omission

Note. Binary-relations are defined over entities, binary-relationships and binary-relations. As such the FACT machine can be used to store certain expressions of higher-order logics. Entity-sets are treated like all other entities. The 'generic rules' such as 'bridges.carry.traffic' may be thought of as domain and counter-domain restriction rules. Negation is represented by omission, although it could be represented as follows:

F#1	. John	. employed by.IBM	
F#2	. F#1	. ε.negative facts	

9. INTERFACING COMPONENTS WHICH HAVE BEEN CHARACTERISED BY SEMANTIC CONCEPTS

Suppose that we are wanting to use the FACT machine as a back-end mass storage structure interfaced to a PROLOG front-end. From the characterisations given above, we can see that:

(*a*) FACT would only be able to readily store a subset of the knowledge which can be expressed in PROLOG. In particular, the basic unit of knowledge which can be represented in FACT is a binary-relationship. In PROLOG terminology this is a unit assertion clause consisting of a two-place predicate. The generic rules in FACT such as 'bridges.carry.traffic' are equivalent to sets of rules expressed in PROLOG, such as the following, depending on what interpretation of the rule is required:

```
carry (X, Y)  :-bridge (X), traffic (Y)
traffic (Y)   :-bridge (X), carry (X, Y)
bridge (X)    :-traffic (Y), carry (X, Y).
```

Thus FACT could be readily used to store unit assertion clauses and sets of PROLOG rules such as the set above,

(*b*) the FACT machine could be used to store some of the more complex rules which can be expressed in PROLOG (and some rules which cannot be expressed in PROLOG) using the fact label, e.g.

F#1 X	. has	. two wheels
F#2 X	. has	. handlebars
F#3 F#1	. and	. F#2
F#4 F#3	. implies	. F#5
F#5 X	. ε	. bicycles

This is equivalent to the PROLOG clause:

```
bicycle (X) :- has (X, two wheels), has (X, handlebars)
```

From such considerations we would obtain an understanding of the type of interface which would be viable between FACT and PROLOG.

10. CONCLUDING COMMENTS

The examples above indicate that a good deal of work needs to be done in order to identify and adequately define a set of semantic concepts which would be useful for the purpose outlined. In particular, we need to be able to specify types of rule.
In addition we need to be able to describe the types of knowledge manipulation which KBS components provide. The above discussion was only concerned with the types of knowledge which can be readily represented. Of equal importance are the functions which the components are capable of performing. Such functions might include:

insertion
deletion
amendment
consistency checking
inference, etc.

A small working group is currently applying the version (0) set of concepts to a range of knowledge-representation formalisms with the objective of refining and extending the set to produce a version (1) set of well-defined concepts. The members of this group are: E. Babb, ICL Systems Strategy Centre; A. Basden, ICL Corporate Management Services, Decision Support Systems Group; R. Frost, Computer Science, University of Glasgow; R. Scowen, Division of Information Technology and Computing, NPL; Dr. I. Torsun, Computer Studies, University of Leeds.

The knowledge-representation formalisms which we are using as examples include those as used in the following: (*a*) IMS hierarchical DBMS; (*b*) IDMS network DBMS; (*c*) INGRES relational DBMS; (*d*) PROLOG; (*e*) LISP; (*f*) various semantic nets; (*g*) various frame-based systems e.g. KLONE, KRL, LOOPS, UNIT; (*h*) first-order predicate logic (FOPL) expressed as formulas; (*i*) FOPL expressed in clause form; (*j*) temporal logic-the Kt system; (*k*) belief logic as used in Konolige;[12] (*l*) production systems as used in expert systems; (*m*) implication network as described in Babb;[10] (*n*) the FACT machine;[3] (*o*) the intelligent file store IFS.[2] The version (1) set of semantic concepts together with characterisations of the above using the version (1) set is scheduled to be available by mid 1985. The version (1) set will use well-established terminology and definitions of concepts which agree with definitions as given in Marciszewski.[20]

Downloaded from https://academic.oup.com/comjnl/article/28/2/112/343415 by guest on 10 April 2024

REFERENCES

1. E. M. Nijssen, From databases towards knowledge bases. In *Databases – a Technical Comparison*, State of the Art Report, edited P. J. H. King (1984).
2. S. H. Lavington, M. Standring and G. B. Rubner, A 4 Mbyte associative predicate store. *Proceedings of the Workshop on Architectures for Large Knowledge Bases*, Manchester University (1984).
3. D. R. McGregor and J. R. Malone, An integrated high performance, hardware assisted, intelligent database system for large-scale knowledge bases. *Proceedings of the Workshop on Architectures for Large Knowledge Bases*, Manchester University (1984).
4. R. J. B. Taylor, Relation memory. *Proceedings of the Workshop on Architectures for Large Knowledge Bases*, Manchester University (1984).
5. M. Azmoodeh, Automatic maintenance of integrity rules in a binary-relational database. *Proceedings of the Workshop on Architectures for Large Knowledge Bases*, Manchester University (1984).
6. R. A. Frost and S. A. Whittaker, A step towards the automatic maintenance of the semantic integrity of databases. *The Computer Journal* **26** (2), 124–133 (1983).
7. R. Reiter, Deductive Q-A on relational data bases. In *Logic and Data Bases*, edited H. Gallaire and J. Minker. Plenum Press (1978).
8. D. H. D. Warren, Efficient processing of interactive relational database queries expressed in logic. In *Proceedings of the 7the VLDB Conference*, 272–281 (1981).
9. A. G. Cohn, Mechanising a particularly expressive many sorted logic. *Ph.D. Thesis*, Essex University (1983).
10. E. Babb, The logic language PROLOG-M in database technology and intelligent knowledge-based systems. *ICL Technical Journal* (1983).
11. G. Mamdani and Gaines, *Fuzzy Reasoning and its Applications*. Academic Press (1981).
12. K. Konolige, Circumscription ignorance. *Proceedings of the Conference on Artificial Intelligence, AAAI-82*, University of Pittsburgh, PA (1982).
13. R. A. Kowalski, Logic databases. *Proceedings of the Workshop on Architectures for Large Knowledge Bases*, Manchester University (1984).
14. R. G. Johnson and N. J. Martin, Triples as a substructure for more intelligent databases. *Proceedings of the Workshop on Architectures for Large Knowledge Bases*, Manchester University (1984).
15. N. J. Martin, The construction of interfaces to triple based databases. *Proceedings of the British National Conference on Databases*, edited P. Hammersley. Cambridge University Press.
16. T.R. Addis, A relation-based language interpreter for a CAPS. *ACM Transactions on Database Systems* **7** (2) (1982).
17. J. Cowie, Building databases from NL input. *Proceedings of the Conference on Applied Natural Language Analysis*, Santa Monica, CA (1983).
18. K. Sparck Jones and B. K. Boguraev, How to drive a database front end using general semantic information. *Proceedings of the Conference on Applied Natural Language Analysis*, Santa Monica, CA (1983).
19. J. J. van Griethuysen, *Concepts and Terminology for the Conceptual Schema and the Information Base*. Preliminary Report, ISO TC97/SC5/WG5 (1981).
20. W. Marciszewski, *Dictionary of Logic*. Martinus Nÿhoff, The Hague (1981).
21. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. Springer Verlag (1981).

Downloaded from https://academic.oup.com/comjnl/article/28/2/112/343415 by guest on 10 April 2024