Second Edition

ENCYCLOPEDIA OF ARTIFICIAL INTELLIGENCE

Volume 1
A-L

Awarded

American Library Association's

Outstanding Reference Source

Association of American Publishers Award

Best New Professional and Scholarly Publication

Stuart C. Shapiro
Editor-in-Chief

Representation & Reasoning

Bryan Kramer and John Mylopoulos,
KNOWLEDGE REPRESENTATION, 743-759

From Shapiro, Stuart C., Editor-in-Chief,
Encyclopedia of Artificial Intelligence, 2nd
Ed., John Wiley & Sons, Inc., NY, 1992.

KNOWLEDGE REPRESENTATION

The dominant paradigm for building intelligent systems since the early 1970s has been based on the premise that intelligence presupposes knowledge. Thus to build a program that performs, say, diagnosis for an infectious disease, it is necessary to identify the units of knowledge used by human experts as they perform the diagnostic task and to make them available to the system under development. Moreover, it is necessary to characterize the patterns of reasoning used by the human expert (deductive, hypothetical, or heuristic) and endow the intelligent system with analogous reasoning capabilities. This methodology is applicable for systems intended to perform any task requiring intelligence, be it diagnosis, planning, design or interpretation. Generally, knowledge is represented in the system's *knowledge base,* which consists of data structures and programs. In addition, the intelligent system is expected to have a program called an *inference engine* that implements the reasoning patterns necessary for the task at hand. Thus current AI theory and practice dictate that intelligent systems be knowledge based, consistent with this simple knowledge base plus inference engine architecture. This emphasis on knowledge has led

to suggestions that AI can be arguably called applied epistemology. More important, it has placed knowledge representation and reasoning at center stage in AI research activity. Although the notion of knowledge representation may seem straightforward, its task and methods have stirred considerable controversy. Indeed, it has been repeatedly documented in questionnaires, panel discussions, workshops, surveys and reviews of the field that there is little agreement on how the problem is to be solved or even what constitutes a solution (Brachman and Levesque, 1985; Levesque, 1986; Cercone and McCalla, 1987).

This article begins with a short discussion of terms used to describe the contents of a knowledge base and the enterprise of knowledge representation, followed by a brief history of the field. The main body of the article discusses the basic functionality of a knowledge representation system and the trade-offs between different paradigms for representing knowledge. A number of important issues is also discussed, ranging from the expressiveness and the semantics of a notation for representing knowledge to structuring facilities for knowledge bases and different categories of reasoning. Examples of knowledge representation systems are included and some areas of active research are mentioned. It must be noted that the discussion does not cover nonsymbolic representations, such as ones based on neural networks (qv), or analogical ones (Funt, 1980).

Vocabulary

A knowledge base is a symbol structure representing a collection of facts about some domain of discourse. The term *object* is used to refer to elements in the knowledge base that are intended to denote entities in this domain. For example, a knowledge base having ancient Greece as its domain of discourse might contain an object representing the person Socrates. Objects are also used to represent abstract concepts such as the concept of person and sets of entities such as the set of philosophers. Terms such as *concept, constant symbol,* and *frame* are commonly used synonymously.

A *fact* is a statement that certain relationships hold between entities denoted by some objects. For example, it might be stated that Socrates's birthplace was Athens. Such statements about the world are variously called *propositions, assertions, formulas, sentences, clauses,* or *properties of objects* in different knowledge representation schemes. Notations for representing knowledge will be referred to as knowledge representation schemes. Examples of schemes include logical calculi, graph-theoretic notations, and even some programming languages. A knowledge representation system offers facilities for constructing and querying a knowledge base.

History

Philosophy has concerned itself with the nature of knowledge since Aristotle and the formalization of reasoning since Leibniz. Mathematical logic (see INDUCTION, MATHEMATICAL) developed at the turn of the century placed mathematics and mathematical reasoning on a formal foundation and has served as a linguistic basis and methodological paradigm for much of the work on the field of knowledge representation (McCarthy, 1968; Hayes, 1977).

An edited volume (Minsky, 1968) contains one of the first strong arguments for the advantages of knowledge-based accounts of intelligence. It includes a collection of papers describing a first generation of knowledge-based programs, many of which argue that access to an organized body of knowledge is necessary for programs to find solutions to commonsense problems in a reasonable amount of time. The collection includes a paper by McCarthy, originally published in 1958, which offers an account of how the program Advice Taker could use knowledge about a task, represented in terms of logical formulas, and could even improve its performance through external advice. This work prescribes much of the research on knowledge representation and reasoning that has unfolded since, although McCarthy's original conception of the Advice Taker remains ambitious even by today's standards. Another important paper in this collection, authored by Quillian, introduces the notion of semantic network (qv), a graph-theoretic data structure whose nodes represent word senses and whose arcs express binary semantic relationships between these word senses. This approach to representing knowledge has been adopted by many subsequent knowledge representation systems. Moreover, Quillian's work constitutes an early attempt to offer a computational account of the associative features of human memory, including a spreading activation model of computation that has served as basis for more recent work on massively parallel machine architectures.

The straightforward application of mathematical logic to knowledge representation, originally attempted in the late 1960s, encountered severe difficulties due to a number of factors. First, general-purpose theorem proving (qv) is computationally intractable, even with the advances in efficiency due to the introduction of the resolution principle. Second, logic, because of its initial focus on mathematics and mathematical reasoning, offered little for the type of commonsense, nondeductive reasoning required of many tasks. An early expression of this difficulty is the so-called frame problem: to reason about actions, such as picking up a book from a table, logical representations require explicit representation of the myriad facts that do not change by the action, eg, the location of the table or the agent's car, in addition to those that do. As a reaction to these difficulties, a number of procedural representation schemes were introduced in the early 1970s. These combined the features of programming languages such as LISP (qv) with new associative data structures and event-driven procedure invocations. Notable among these were PLANNER (qv) (Hewitt, 1971) and its successor, CONNIVER (qv) (Sussman and McDermott, 1972).

Production systems constitute another proposal for representing knowledge procedurally, offered during the same period (Newell and Simon, 1972). In a production system, knowledge is expressed in terms of if–then rules, which specify condition–action or premise–conclusion pairs (see RULE-BASED SYSTEMS). For example, a rule might

state "*if* the top of a block to be moved is not clear, *then* clear it!" This scheme is the basic representation of knowledge used by most early expert systems (qv) (Hayes-Roth and co-workers, 1983). An early and authoritative collection of papers on production systems has been published (Waterman and Hayes-Roth, 1978).

As semantic network representation schemes proliferated in the 1970s, it became clear that there was no consensus on their form or their semantics (see SEMANTIC NETWORKS). A paper by Woods (1975) notes this difficulty and suggests possible solutions toward a well-founded semantics. This influential paper spawned a new generation of semantic network schemes (Findler, 1979), including schemes that treat semantic networks as a graph-theoretic notation for logical formulas.

Also during the mid-1970s, Minsky (1975) introduced the notion of *frames* (qv) as a means for representing commonsense knowledge, such as the concept of a room or an elephant. A frame is a complex data structure containing information about the components of the concept being described, links to similar concepts, as well as procedural information on how the frame can change over time. Examples of early frame-based representations are KRL (qv) (Bobrow and Windograd, 1977), FRL (qv) (Goldstein and Roberts, 1977), and PSN (Levesque and Mylopoulos, 1979).

Since the early 1980s there have been attempts to offer knowledge representation schemes that adopt and integrate ingredients from logic, semantic networks, and procedural representations. An example of this new trend is KRYPTON (Brachman and co-workers, 1983). A KRYPTON knowledge base consists of two components: a semantic network-based component where terms are described and a logic-based one, including assertions about the domain of discourse. For example, a KRYPTON knowledge base may include a description for the term "bachelor", defined as an unmarried male person, along with an assertion involving "bachelor", for example, "John is a bachelor."

KNOWLEDGE REPRESENTATION SYSTEMS

At the very least, it would be expected that a knowledge representation system would provide functions for incrementally constructing and for querying a knowledge base. These might be defined as follows:

$$\text{Tell } [KB, \alpha] = KB'$$

where KB' is the result of adding fact α to KB,

$$\text{Ask } [KB, \alpha] = \text{yes} \mid \text{no} \mid \text{unknown}$$

depending on question α and the contents of KB. Underlying these operations is a notation for representing facts and queries. Proposed notations will be classified into three basic paradigms: logic-based, procedural, and semantic network.

Logic-Based Representations

Adopting the notation of mathematical logic, one might express facts such as

> Socrates is human.
> If someone is human, then she is mortal.

with formulas such as

$$\text{human (socrates)}$$

$$(\forall X)\,[\text{human}\,(X) \Rightarrow \text{mortal}\,(X)]$$

A knowledge base can then be viewed as a collection of such formulas and addition of a new fact amounts to an extension of the knowledge base to include another formula:

$$\text{Tell } [KB, \alpha] = KB \cup \alpha$$

From this perspective, querying the knowledge base could be defined semantically or syntactically, using the notions of (logical) consequence and provability from mathematical logic. In particular, if α is a closed formula, representing a yes–no question, then a semantic account of the query operation would be

$$\text{Ask } [KB, \alpha] = \begin{cases} \text{yes} & \text{if } KB \models \alpha \\ \text{no} & \text{if } KB \models \neg\alpha \\ \text{unknown} & \text{otherwise} \end{cases}$$

In other words, the answer to α is yes (respectively no) if α is true (respectively false) in all interpretations (or worlds) where all formulas in the KB are true. A syntactic account of the query operation, on the other hand, is

$$\text{Ask } [KB, \alpha] = \begin{cases} \text{yes} & \text{if } KB \vdash \alpha \\ \text{no} & \text{if } KB \vdash \neg\alpha \\ \text{unknown} & \text{otherwise} \end{cases}$$

In this case, the answer to α is yes if α can be proved from the facts in the knowledge base using the provability relation associated with the underlying logic. A major advantage of many logics adopted for knowledge representation is that they are sound and complete, which means that derivability and provability lead to the same set of consequences, given a knowledge base.

Choosing a logic that is both expressively adequate for knowledge representation and computationally tractable has turned out to be an elusive goal. Propositional logic is generally considered expressively inadequate, whereas first-order logic is generally conceded to be computationally intractable. Attempts to find an acceptable compromise to the expressiveness versus tractability trade-off generally use variations of first-order logic, following one of two approaches. The first approach limits the expressiveness of the language of representation by restricting the form of the formulas that can be admitted in the knowledge base. The second approach redefines the prov-

ability relation of first-order logic to make it computationally tractable.

Relational databases (Date, 1975), widely used to represent "simple" facts, such as people's addresses or salaries, constitute a good example of the first approach. A relational database can be viewed as a collection of ground formulas of the form $P(a_1,a_2, \ldots ,a_n)$, where P is a predicate symbol and $a_1a_2, \ldots ,a_n$ are constant symbols. Accordingly, the language of representation (determined by *Tell*) disallows logical variables, connectives, and quantifiers. This restriction makes it impossible to explicitly represent negative knowledge and implications. The query language (determined by *Ask*) does, however, maintain the full power of first-order logic. It should be mentioned that relational databases have a number of built-in axioms, including the so-called closed world assumption (Reiter, 1984), which states that ground facts not present in the database may be assumed to be false. Thanks to the closed world assumption, negative information is represented in the database, albeit implicitly. Computationally, it can be easily established that the complexity of *Ask* and *Tell* depends polynomially on the size of the knowledge base.

Logic programming (qv) constitutes a second popular approach to harnessing the expressiveness of first-order logic. Logic programs admit only two types of formulas: *atomic clauses,* having the form of positive literals, ie, formulas $P(t_1,t_2, \ldots ,t_n)$ where $t_1,t_2, \ldots ,t_n$ are terms involving function symbols, constants, and (universally quantified) variables, and *horn clauses,* having the form $A_1 \wedge A_2 \wedge \ldots \wedge A_m \Rightarrow B$, where $A_1,A_2, \ldots ,A_m$ and B are positive literals. Here queries are restricted to conjunctions of positive literals known as goals. Evaluation of a query is equivalent to proving the conjunction with respect to the clauses in the logic program. The restrictions to the assertion and query languages allow the use of specialized theorem provers. For example, PROLOG (Kowalski, 1979) uses a particular form of resolution called SL resolution, which always starts from the goal to be proved and reduces it, using Horn clauses, to a set of atomic clauses.

Logic programming supports a declarative reading of the clauses constituting a logic program, ie, a reading where each clause is treated as a true statement about the intended domain of discourse. However, it also supports a procedural reading, treating a Horn clause such as

$$A_1 \wedge A_2 \wedge \ldots \wedge A_m \Rightarrow B$$

as the statement "to establish B, establish $A_1,A_2, \ldots ,A_m$." Indeed, in programming languages such as PROLOG that support the logic programming paradigm explicit facilities promoting this procedural interpretation of clauses are provided. For instance, the cut operator limits the search that will be undertaken by a PROLOG interpreter for a particular goal.

Like logic programming, deductive databases attempt to establish a computationally tractable class of knowledge-representation systems grounded in logic. Deductive databases impose different types of restriction on the form of allowable formulas. For example, definite deductive databases allow Horn and atomic clauses, analogous to those of logic programs. Hierarchic databases, on the other hand, restrict Horn clauses to be nonrecursive, thereby simplifying goal evaluation. Efficient algorithms have been developed for goal evaluation and consistency checking of definite deductive databases (Hulin and co-workers, 1989). Deductive databases differ from logic programming in that they assume that the number of atomic clauses in a knowledge base will be much greater than that for nonatomic ones. Moreover, deductive databases emphasize the declarative reading of clauses and rely completely on the efficiency of the adopted goal evaluation and consistency checking algorithms.

Turning to approaches that redefine the provability relation in order to render a logical representation computationally tractable, a logic has been offered that uses the full expressive power of first-order logic but interprets $(\exists X)\, P\,(X)$ to mean something like "there exists a known X such that $P\ (X)$" (Frisch and Allen, 1982; Patel-Schneider, 1985). This apparently slight change in the semantics of existential quantification has a remarkable impact on the provability relation. In particular, *modus ponens* no longer applies as inference rule, and from $P\ (a) \vee P\ (b)$ it cannot be inferred that $(\exists X)\, P\ (X)$.

Procedural Representations

Declarative representations treat the intended meaning of a knowledge base as a foundation that imposes constraints on knowledge base operations. Procedural representations, on the other hand, reverse this dependence by identifying the meaning of a knowledge base with its use.

Consider a knowledge base consisting of logical formulas:

$$(\forall X)\ \text{person}\ (X) \Rightarrow \text{mortal}\ (X)$$

$$(\forall X)\ \text{dog}\ (X) \Rightarrow \text{mortal}\ (X)$$

$$\text{person (socrates)}$$

$$\text{person (helen)}$$

which states that "all persons are mortal," "all dogs are mortal," "Socrates is a person," and "Helen is a person." Given an inference procedure for first-order logic, it is possible to find out whether Socrates is mortal by attempting to prove the formula *mortal (socrates)* from the facts in the knowledge base. If the inference procedure is sound, inferred facts are true in every interpretation where the facts in the knowledge base are true. A great strength of declarative representations is that the same inference procedure can be used for any knowledge base and any knowledge based system, independently of the domain of discourse and the task to be performed by the system.

A procedural representation, in its purest form, treats the knowledge base creator as a programmer who must decide on a set of data structures and programs that represent the intended facts and generate appropriate inferences. The meaning of the knowledge base is now determined by the data structures and programs that implement it. For example, the same knowledge base

above might be represented in terms of a collection of procedures, such as the procedure *person:*

procedure person (X)

if (X = 'socrates') **or** (X = 'helen') **then** return (**true**)

else return (**false**)

which returns true or false, depending on whether its argument is a person or not, or the procedure *mortal,* which answers the corresponding question for the predicate mortal:

procedure mortal (X)

if person (X) **then** return (**true**)

else if dog (X) **then** return (**true**)

else return (**false**)

Here it could be asked whether Socrates is a person by executing *person* ('*socrates*') and whether Socrates is mortal by executing *mortal* ('*socrates*'). In this case, the facts constituting the knowledge have been hard coded into the procedures that define the inference engine of the knowledge base.

Clearly, the declarative approach has advantages of flexibility and modularity. First, declarative representations, including logic programming, allow queries such as *mortal* (X), where X is a variable. The variable X is interpreted in this query existentially and the inference procedure can return all bindings of X that make the clause true with respect to a knowledge base. This facility is possible in the procedural representation but requires foresight on the part of the knowledge base creator who much define a separate procedure, say *AllMortals* (), that returns a list of all mortals. Note that the list returned by the new procedure may not be consistent with the rest of the procedures that define the knowledge base.

The advantages of declarative representations are also obvious when a fact is inserted or removed. For example, adding the fact that horses are mortal,

$$(\forall X)\ \text{horse}\ (X) \Rightarrow \text{mortal}\ (X)$$

to the declarative knowledge base is quite simple. In the procedural case it would be necessary to modify several existing procedures to effect this change.

The main advantage of the procedural representation lies in the control of search that can be exercised by the knowledge base creator, given that the inference engine is defined for each particular knowledge base and each usage of that knowledge base. In the above simple example, the procedure *mortal* (X) first checks whether X is a person and then whether X is a dog. This choice of alternatives may be based on the programmer's knowledge that the argument of this procedure is more often a person than a dog. In a knowledge base using a declarative representation, on the other hand, there is no room for such pragmatic consideration in the conduct of search by the inference procedure. Note, however, that meta-level control (discussed below) is an attempt to exercise precisely this kind of control within a declarative setting.

Some of the drawbacks of procedural representations can be overcome by generalizing procedural mechanisms such as those found in conventional programming languages. One such proposal concerns the generalization of the mechanism for invoking procedures so that neither the procedure's name nor its arguments need be provided explicitly in a procedure call. Thus instead of writing *mortal* ('*socrates*') to invoke the procedure called *mortal* with the string '*socrates*' as its argument, *goal* (*mortal* ('*socrates*')) could be written to specify "run any procedure whose description states that it might be able to find out if Socrates is mortal." For this invocation mechanism to work, procedures must have an associated pattern that specifies what the procedure can accomplish and that is matched against the argument of a goal statement to determine whether the procedure can help. This kind of procedure invocation is known as pattern-directed invocation and is clearly akin to the procedural interpretation of Horn clauses.

Procedural representations that support pattern-directed invocation generally allow more than one procedure to be called during the evaluation of a goal expression. If a procedure is unable to provide an answer (because it fails during its execution) others are tried, with the knowledge base restored to its original state. This search regime is called backtracking (qv) and is also used by PROLOG interpreters. However, by incorporating pattern-directed invocation and backtracking into a system, some control of search has been given up.

Another approach to overcoming some of the drawbacks of procedural representations is to combine them with declarative ones. For instance, some authors have proposed knowledge representation schemes that contain both a declarative and a procedural component. In knowledge bases using this representation scheme, some inferences are done by using the inference procedure that comes with the declarative representation. However, under certain conditions, inference is performed by externally defined procedures contained within the knowledge base. For example, it might be specified that the successor property of numbers should be computed by the LISP function ADD1 rather than through inference. Likewise, procedural representations have been combined with semantic network ones leading to frame-based and object-oriented representations (see below). The language PLANNER (qv) and production systems are generally viewed as pioneering proposals for procedural representations.

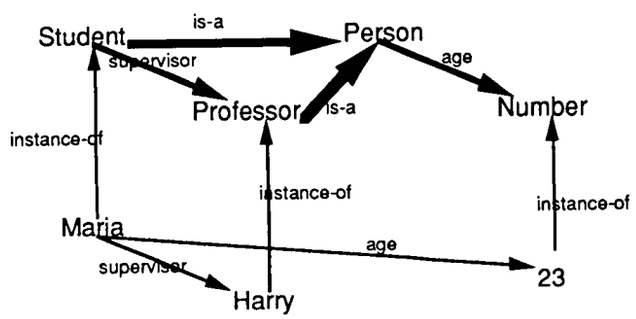
Semantic Networks

As indicated earlier, semantic networks (qv) (Sowa, 1991) were originally motivated by cognitive models of human memory. However, their popularity and success can best be understood when they are viewed as a convenient compromise between the declarative and procedural extremes. Self-styled semantic network proposals for knowledge representation abound in the literature. One striking feature of this collection is the small number of

common themes and the great number of variations that can be found among them. One such theme is that knowledge is represented on a labeled, directed graph whose nodes represent concepts and entities in the domain of discourse, while its arcs represent relationships between these entities and concepts. For example, a knowledge base might be viewed as in Figure 1. In Figure 1*a*, the nodes represent generic concepts such as *Person, Student, Professor,* and *Number,* or particular entities such as *23, Maria,* and *Harry*; the arcs represent binary relationships, such as *supervisor, age, instance-of,* and *is-a*. This depiction can be thought of as a convenient data structure for representing the knowledge base displayed in Figure 1*b*.

In general, it has been argued (Shapiro, 1971; Schubert, 1976) that semantic networks simply offer a graphical notation for logical formulas restricted to unary and binary predicates and the two types of quantification illustrated in the example (declaring, respectively, that some concept "is-a" other and that some concept is "part-of" another). Others have argued that semantic networks offer a fundamentally different representational paradigm that is object centered in the sense that it is based on object descriptions rather than arbitrary propositions, and focuses on knowledge organization. Advantages claimed for this paradigm include efficient retrieval, because all facts about Maria, for instance, are directly accessible from the node named *Maria,* as well as the obvious perspicuity of graphic representations.

An interpreter for such a semantic network would be expected to support some inferences, for example, the transitivity of is-a and the typing constraints of part-of ("ages must be numbers and supervisors professors"). However, beyond these, opinions differ on how to represent other kinds of facts involving disjunction, negation, and unconstrained quantification. One approach to this dilemma involves offering a graph-theoretic representation of general logical formulas. This allows the inference procedure to perform retrieval and certain inferences (eg, ones involving *is-a* and *part-of*) efficiently, whereas others are handled using general theorem proving techniques. A second approach allows the attachment of procedures to each node that specify operations on instances of that node and support "local" inferences with respect to the concept being represented by the node. The first approach overcomes some of the drawbacks of logical representations because special-purpose inference procedures can be used for the "built-in" features of semantic network representation. The second approach removes some of the *ad hoc* nature of procedural representation because the representation fixes the data structure and offers a method for defining the procedures that constitute the knowledge base. There are interesting similarities between the second approach and object-oriented programming systems (Keene, 1989).



(a)

(∀X) student (X) ⇒ person (X)
(∀X) professor (X) ⇒ person (X)
(∀X) person (X) ⇒ (∃Y) [number (Y) ∧ age (X,Y)]
(∀X) student (X) ⇒ (∃Y) [professor (Y) ∧ supervisor (X,Y)]
student (maria)
professor (harry)
number (23)

(b)

Figure 1. A semantic network proposal for knowledge representation. A labeled, directed graph (*a*) is used to represent a knowledge base (*b*).

A major feature of semantic networks since Quillian's proposal was the idea of default inheritance, according to which attributes associated with a node, say *Person,* are only inherited by its *is-a* descendents, such as *Student* and *Professor,* provided they are not redefined explicitly (think of ageless mythical persons or gifted students supervised by committee or by no one at all!). Default inheritance simply acknowledges the fact that descriptions of concepts are not absolute and that they may be contradicted by their specializations or their instances. This aspect of semantic networks is not captured in the simple logical reformulation of semantic networks suggested above. Indeed, default inheritance constitutes an important example of commonsense reasoning that cannot be addressed adequately within the framework of traditional mathematical logic. Minsky's proposal of a frame theory (qv) can be seen as an attempt to extend and refine the semantic network paradigm by adopting a particular stand on what is represented by a node (a frame in Minsky's terminology). A frame is a complex data structure representing a prototypical concept or situation. For example,

Frame	Myrto		
	AKO	**$value**	Person
	name	**$value**	|Myrto Wong|
	address	**$value**	|65 Elm Street|
	interests	**$value**	'AI

defines the frame identified as *Myrto* and declares it to be *a kind of* (hence AKO) *Person,* with particular name, address, and interests. A frame consists of slots that define its constituent parts. Each slot has a number of facets that specify local information such as the slot's value, as well as inferences and consistency checks that should be performed when the slot value is retrieved or updated. Adopting the principles of default inheritance, all slots are inherited along *AKO* links by default. For example, if the frame *Person* includes the following in its definition,

```
frame Person
  . . .
  state        $require    [member (:value, States) or
                            (member (:value, StateAbbr)]
  . . .
  age          $type       Number
  yearOfBirth  $value              !calcYearOfBirth (:age)
  . . .
  salary       $ifAdded    [if (:value > $100,000)
                            then add (:frame, VIPList)]
```

Myrto inherits a *state* slot with a constraint that its value facet be a member of the list *States,* a *yearOfBirth* slot whose value is computed by the expression *calcYearOfBirth* (*:age*) where *:age* refers to the *age* slot of *Person,* and a *salary* slot that, when updated, will cause *Myrto* to be added to the list *VIPList* if the person's *salary* value is greater than $100,000.

Frame representations focus on the problem of representing prototypical concepts. Terminological languages, on the other hand, offer facilities for the definition of terms that might be used as vocabulary in expressing facts about a given application. To build a knowledge base about persons, for example, it is possible to start with primitive terms such as *Person, MaritalStatus,* and *Sex* (which are predicates of one, two, and two arguments, respectively) and then proceed to define other terms such as *Unmarried, Man,* and *Bachelor* as follows:

Unmarried ← (VRgeneric Person MaritalStatus single)

Man ← (VRgeneric Person Sex male)

Bachelor ← (Congeneric Man Unmarried)

VRgeneric and *Congeneric* are term-forming operators; *VRgeneric* defines a term from another by restricting a slot value and *Congeneric* defines a term as a conjunction of two terms. Thus *Unmarried* is defined as a person whose marital status is single, *Man* as a person whose sex is male, and *Bachelor* as someone who is both a man and unmarried. Subsumption is another important operation on terms, intended to compare two terms and determine whether one is more specialized than another. For instance, the term *Man* subsumes the term (*Congeneric* (*VRgeneric Person Sex male*) *Unmarried*) in the sense that every instance of the latter is also an instance of the former. Much recent research has focused on the trade-offs between the set of term-forming operators provided by a terminological notation and the tractability of subsumption for that notation (Brachman and Levesque, 1985). KL-ONE (Brachman, 1979) has played a key role in shaping terminological languages and raising a host of research issues related to their design.

Knowledge Organization

Both the semantic network and the frame examples given above adopt primitives for organizing knowledge. In the case of the semantic network example, these primitives are *is-a* and *instance-of* links. In the case of the frame example, they include *AKO* links as well as the slot mechanism. This is not an accident. Knowledge organization constitutes a major strength of semantic network and frame-based representations.

Organizing the knowledge included in a knowledge base is important from a cognitive science viewpoint [after all, human memory seems to be highly structured (Anderson, 1990)] as well as for reasons of computational tractability. Human factors considerations also contribute to the importance of knowledge organization. Both developers and users of a large knowledge base will find it easier to understand its contents if it can be visualized as a structured artifact. In this respect, an analogy can be drawn between knowledge bases and programs. Abstraction mechanisms and other means for structuring programs have been accepted for some time as essential tools for addressing the complexities of understanding, using and maintaining large programs.

A review of structuring mechanisms that have been adopted for knowledge organization finds remarkable consensus based on a few themes, but there is also a large number of variations.

Instance-of (or Classification). This structuring mechanism classifies an object into one or more generic classes (makes the object an instance-of these classes), thereby endowing it with properties shared by similar objects. For example, some objects in a knowledge base might be classified under *Person* and can, therefore, have an address and an age, whereas others are classified under *Dog* and, therefore, possess four legs.

Classification has been used in a variety of notations to support syntactic and semantic consistency. For example, sorts in mathematical logic (Cohn, 1989) and types in programming languages are used mostly for syntactic checking. In semantic networks, classification distinguishes between tokens, which represent particular individuals in the domain of discourse, and types or classes, which represent generic concepts. Besides syntactic and semantic consistency, classification can also lead to more efficient search algorithms for a knowledge base. If, for instance, the system is looking for an object whose student number is 98765432 and it is known that only students have student numbers, then only the set of instances of *Student* must be searched.

Some representation schemes (eg, PSN) allow classification to be recursive, ie, classes may (or must) themselves be instances of other classes. In this case the class *Person* might be an instance of the (meta)class *AnimateClass,* which has as instances all classes describing animate objects. In such situations classification may be unconstrained, allowing both a token and a class that it is an instance-of to be instances of some common class, or constrained in the sense that there is a linear order of strata (or levels) and that every object in the knowledge base belongs to a unique stratum and that an object at stratum τ can only be an instance of classes at stratum $\tau + 1$.

Is-a (or Generalization). Generic objects in a knowledge base have been traditionally organized into taxonomies, referred to as is-a, or generalization, hierarchies, which organize all classes in terms of a partial order relation

determined by their generality or specificity. For example, *GradStudent* may be declared as a specialization of *Student* ("Every graduate student is a student"), which is in turn a specialization of *Person* ("Every student is a person").

Inheritance is a fundamental ingredient of is-a hierarchies. Inheritance is an inference rule that states that attributes and properties of a class are also attributes and properties of its is-a descendants. Thus the *address* and *age* attributes of *Person,* are inherited by *Student* and, transitively, by *GradStudent.* This inheritance may be strict in the sense that constraints on attributes and properties can be strengthened but cannot be overridden or defaulted, in which case overriding is allowed. For example, if the age of persons has been declared to range from 0 to 100 years, with strict inheritance the age of students can be declared to range from 5 to 80 but not from 5 to 120. Default inheritance, on the other hand, allows students to be 120 years old, although persons were declared to live only up to 100 years, or penguins to not fly although birds were declared to do so.

Another source of variation among generalization mechanisms is their form. In single inheritance, every class has at most one immediate is-a ancestor, hence the graph formed by classes and is-a relationships is a tree. In multiple inheritance, a class is allowed to have multiple immediate is-a ancestors. Single inheritance is computationally and conceptually preferrable but expressively inadequate for sophisticated applications. Multiple inheritance, on the other hand, is expressively more powerful but also computationally expensive and semantically troublesome, particularly when combined with default inheritance.

Part-of (or Aggregation). This mechanism views objects as aggregates of their components or parts. Thus a person can be viewed as a (physical) aggregate of a set of body parts (arms, legs, head and the like) or as a (social) aggregate of a name, address, social security number, etc. Components of an object might themselves be aggregates of yet other simpler components. For example, the address of a person might be declared as the aggregation of a street number, street name, city, etc. Aggregation may be strictly hierarchical or recursive. For instance, *Employee* may be defined as the aggregation of a department, a salary, and an employee, where the latter defines the role of the employee's manager.

A fourth way of organizing objects in a knowledge base is through partitions (Hendrix, 1979) or contexts. A partition is simply a subset of the objects and facts in the knowledge base. One use of a partition is to group together all of the facts about a given situation, eg, the case history of a particular patient. Here partitioning is a useful mechanism for limiting search for relevant facts. A second use of partitions involves the representation of different possible worlds (say, this world versus the world that would have resulted if Napoleon had won the battle of Waterloo) or the representation of facts believed by different agents known to the system. Partitions are usually organized into a hierarchy by a relationship that might be called "obtained from." How a partition B is formed from a partition A is specified by stating what objects and facts have been added to or deleted from A to obtain B. An object that belongs to a partition is said to be visible in that partition. In particular, all objects in A that were not removed in forming B will be visible in B.

Inference. For *Ask* to return a correct answer with respect to a given query and a knowledge base, it must go beyond the facts contained in the knowledge base and draw conclusions from these facts. Inference procedures provide knowledge representation systems with this capability. In general, an inference procedure is a multistep process. Each step of the process involves an inference rule, a set of premises and, if the premises match the rule, a set of conclusions. Returning to an earlier example, if a knowledge base contains

Socrates is human.
If someone is human then he is mortal.

then an inference rule can be applied to these premises to conclude that Socrates is mortal:

Socrates is human.
If someone is human then he is mortal.
Socrates is mortal.

This inference makes use of *modus ponens,* an inference rule that is sound in the sense that in any situation where its premises are true, its conclusions will also be true. Inference based on sound inference rules is known as deductive inference. Frequently, however, conclusions must be drawn on the basis of their plausibility because information in the knowledge base is incomplete. For example, concluding that all ravens are black from the fact that all ravens observed so far are black involves inductive inference, an important form of learning. Likewise, hypothesizing that a patient has a cold after observing that he or she has a running nose involves a form of abductive inference, which uses as premise a fact such as "If someone has a cold then he has a running nose" and an inference rule that drives the implication backward, unlike the deductive inference rule (*modus ponens*) illustrated above. Analogical inference is a form of reasoning where similarities between situations is used as a basis for drawing conclusions. For example, on the basis of the fact that Charles De Gaul was president of France and was more than six feet tall, and the fact that Francois Mitterand is also president of France, it might be concluded, (erroneously) that Francois Mitterand is more than six feet tall. Probabilistic inference involves reasoning on the basis of probabilistic information, in quantitative or qualitative terms. Default inference involves drawing conclusions in the absence of information. These and other forms of inference have been found useful in endowing a knowledge representation system with a reasoning capability and have been formally studied.

Inference procedures are needed independently of the knowledge representation paradigm adopted for a particular knowledge representation scheme. However, these procedures are quite different, depending on the underly-

ing paradigm. Inference procedures for logic-based paradigms are founded on few inference rules (often some variation of the resolution principle) that are independent of the knowledge base and the task being performed by the system using the knowledge base. Procedural representations offer tools for defining both the inference rules and the search strategy used by an inference procedure. This means that both can be tailored to a particular knowledge base and a particular task, say diagnosis or design, if the designer of the knowledge base so desires. For semantic network representations, on the other hand, a limited inference procedure is required for the axioms that underlie is-a or part-of links. Beyond this, however, *Ask* behaves like a retrieval procedure defined for a graph data structure.

The canonical example of an (deductive) inference procedure is a theorem prover for predicate logic (Stickel, 1982). A second, very different, example is an inference mechanism that might be called spreading activation introduced by Quillian. The idea here is that certain problems, such as interpreting the phrase *dog food,* can be solved by searching for paths connecting particular nodes in a semantic network, say

$$\text{dog} \rightarrow (\text{eats}) \rightarrow \text{food}$$

In general there are several paths, for example

$$\text{dog} \rightarrow (\text{made-of}) \rightarrow \text{meat} \rightarrow (\text{a-kind-of}) \rightarrow \text{food}$$

In these examples parentheses are used to indicate edge labels. Moreover, these paths can be discovered by an algorithm that starts from the nodes that define the path endpoints and proceeds by iteratively propagating markers to neighbouring nodes. Such marker passing mechanisms have been adopted in Fahlman's (1979) NETL system. A third example of an inference procedure involves the subsumption operation supported by terminological languages. At the heart of any inference procedure is found some form of matching, used to select facts that will be used as premises in an inference step, and a control structure, which determines the sequence of inference steps that will be used to evaluate a query.

Matching. Matching involves the comparison of a pattern with an object in the knowledge base. The match succeeds if the object is unstructured and the pattern primitively matches the object or if the object is structured and each part of the pattern successfully matches one or more parts of the object. For example, if objects are either letters or lists of letters, the pattern (*A B C*) would match the object (*A B C*).

Pattern matching would be uninteresting if a pattern only matched itself. What makes it powerful are the various ways of specifying a range of objects that can match the pattern. In the above example a pattern element that matches any letter might be allowed. For example, (*A ? C*) could be a pattern that matches any list of three letters of which the first is in *A* and the last a *C*. Another common mechanism is a notation that matches more than one element. The pattern (*A * C*), for example, might match a list starting with an *A*, ending with a *C*, and having any number of intermediate elements. A large variety of mechanisms are possible, and the choice depends on the representation language. There is also a trade-off between the complexity of the pattern and the computational efficiency of the pattern matcher.

A third important ingredient of pattern matching is the ability to associate variables with pattern elements. When a match succeeds, the values of these variables are the parts of the object that matched the corresponding pattern elements. For example, the pattern (*A *x C*) might match any list beginning with *A*, ending with *C*, and having any number of elements in between, and that when the match succeeds, the value of *x* will be the list of elements in between. Thus after matching (*A W X Y C*), the value of *x* would be (*W X Y*).

When both the pattern and the object may contain variables corresponding to pattern elements that can match any object element, the resulting operation is a form of unification. In effect, there is no distinction between pattern and object. This was introduced as part of the resolution methods for proving theorems in logic (Robinson, 1979) (see THEOREM PROVING). In this case, both the pattern and object can be viewed as general expressions including variables. The result of unification is the most general substitution of variables that makes the pattern and object identical. Unification has in the past been used in matching linear expressions: recent work has extended unification to frame-based structures (Ait-Kaci, 1983).

The above discussion focused on a kind of matching where the parts of a structured pattern are put into correspondence with the parts of an object having the same structure. This is known as syntactic matching. Semantic matching, on the other hand, involves finding a correspondence between the pattern and object based on some description of the function or role of the parts in an object. For example, a pattern might specify that a successful match would be a set of two people, one of whom is the husband in a marriage and the other the wife in the same marriage; this might be written as *x*: (*a husband*), *y*: (*the wife of x*).

Various knowledge representation notations make different uses of matching. First, in pattern-directed procedure invocation, such as that found in PLANNER (qv), the pattern associated with a procedure is matched with the current object of interest to see whether that procedure is relevant. Matching is similarly used in production systems to determine whether a rule is eligible for firing. Second, unification plays a fundamental role in resolution-based inference procedures. A third use of matching involves determining the type of an unknown input by treating the input as object and the types under which it could be classified as patterns. The KL-ONE (qv) classifier is an example of this kind of matching.

Control Structures. When a procedural representation makes use of pattern-directed invocation, it is often the case that more than one procedure matches the pattern. One aspect of the control structure determines how such conflicts are resolved. A second aspect of the control structure involves the direction of inference, ie, the mechanism

that chooses the patterns used to select procedures. In the past, the mechanism for conflict resolution has usually been wired into the inference engine. Examples of strategies that have been used include choosing the procedure appearing first in the knowledge base or choosing the procedure with the most complicated pattern. More recently, there has been research into the use of an inference engine using control knowledge to resolve conflicts (Davis, 1980; Genesereth, 1983; Kramer, 1986). This has the advantage that new knowledge about resolving conflicts can be added without the need to recode the inference engine.

An inference engine may use two possible directions to generate inferences. Forward-chaining inference arises when procedures are invoked if their patterns match facts about to be added to or removed from the knowledge base. Generally, such procedures will add and remove facts of their own thus triggering yet other forwardly chained inferences. In contrast, backward-chaining inference works by matching rule patterns to goals. Trying to match a given pattern against the knowledge base may trigger procedures that set other pattern-matching goals in an attempt to achieve the original goal (see PROCESSING, BOTTOM-UP AND TOP-DOWN).

EXAMPLES OF KNOWLEDGE REPRESENTATION SYSTEMS

Planner. PLANNER (qv) is a procedural language extending LISP with a number of mechanisms, including pattern-directed invocation of procedures, an associative database and a backtracking control regime. A system written in PLANNER consists of a database of assertions; these are simply LISP data structures. For example, the fact that Socrates is a man might be represented with the assertion (*MAN SOCRATES*). In addition to the database of assertions a set of procedures can be defined, called theorems, or demons. A theorem is executed when its pattern matches the argument of a database operation. There are two basic kinds of theorems. Antecedent theorems are invoked when their patterns match structures that are being added to the database and correspond to a forward-chaining control structure. Consequent theorems, on the other hand, are invoked in response to a goal statement and, therefore, correspond to backward chaining. A goal statement is used to retrieve information from the database; when the pattern of a goal matches the pattern of a consequent theorem, the theorem is used to derive implicit knowledge matching the goal pattern. The body of the theorem can assert or remove facts or try to prove other goals. For example, a consequent theorem for solving the goal (*MORTAL* ?*x*) might try to prove the goal (*MAN* ?*x*). This theorem will fail if it cannot establish this goal. Because more than one theorem might match a goal pattern, if one fails, the system automatically tries another. Note that when this backtracking occurs, it is necessary to undo the effect of all changes to the database resulting from the body of a theorem to the point of its failure. There are obvious analogies between PLANNER theorems and PROLOG Horn clauses, pattern-directed procedure invocation and clause invocation, and even the backtracking control structure used by both PLANNER and PROLOG. At the same time there are important differences between the two representation schemes. PROLOG offers both a procedural and a declarative semantics and its matching operation, unification, is cleaner and more general than that of PLANNER. PROLOG originated from logic whereas PLANNER was an outgrowth of LISP.

Production Systems. Production systems constitute another class of procedural schemes using pattern-directed invocation. Most commercial expert system shells are based on some form of production system. The basic features of such systems include a global database containing data structures relevant to a computation and a set of production rules that operate on the database. A production rule consists of a procedure called the body of the rule and an associated pattern. Inference is a cycle where the pattern of each rule is matched against the database, a choice is made from the rules whose patterns matched, and the body of the chosen rule(s) is executed. Executing the body of a rule will generally result in changes to the database caused by the addition or deletion of structures. Because rules cannot invoke other rules directly, all communication between rules is done through the database.

There are many sources of variety in production systems. First, the size and complexity of the global database can vary. The early production system proposed by Newell (1973) restricted the contents of the database to a small number of lists. In this work the database was viewed as a model of human short-term memory, and all knowledge stored in long-term memory was to be represented as rules. HEARSAY-II (qv) (Erman and co-workers, 1980), on the other hand, uses a global database called a blackboard, which stores a large number of frames partitioned into layers. Second, production systems can vary in the complexity of the pattern allowed: sometimes the pattern may contain calls to arbitrary functions that decide whether, and how, the pattern matches. Third, the set of actions in the body of rules can vary. Actions can include adding and removing elements from the database, adding and removing rules, and interacting with the user. Fourth, production systems vary in the control structures that are available. When a rule body must always remove the database elements that triggered its evaluation and otherwise may only add elements to the database, the production system can be run using backward-chaining. This control regime has been used by many expert systems.

The final source of variation among production systems is the conflict-resolution mechanism that is used to deal with situations in which more than one rule matches the database. MYCIN (qv) (Shortliffe, 1976), eg, simply executes all of the matching rules. A strict Markovian system, on the other hand, would simply choose the first rule according to some ordering of the rules. Systems such as OPS-5 (qv) (McDermott and Forgy, 1978) have an extensive set of heuristics that use such factors as the complexity of the patterns and the relative ages of competing rules to choose in the case of conflicts. The systems described in Waterman and Hayes-Roth (1978) include a number of additional conflict-resolution schemes. Production sys-

tems share features with both PLANNER and PROLOG. Unlike PLANNER and PROLOG, however, production systems offer a lean representational and control structure framework which can be tailored to particular applications.

SNePS. This is a semantic network-based system that represents a subset of logic in terms of a network structure (Shapiro, 1979). A SNePS network consists of nodes of several kinds: nodes corresponding to constant symbols in logic, nodes corresponding to variables, and rule nodes that correspond to propositions. Links are either representations of binary relations or are auxiliary links that are used to represent the properties of nonconstant nodes. The logical foundations of the representation have been exploited in defining a semantics and an inference mechanism.

KRL. KRL (qv) is perhaps the most ambitious frame-based system among early efforts (Bobrow and Winograd, 1977). The basic representational structure is called a unit. A unit consists of several descriptions, each giving a different perspective on the entity. As prescribed by Minsky, a primary form of description is via further specification of a prototype where the prototype is a typical member of a category. Other kinds of description include description of the role played by an object, a logical formula that is true of the entity, membership in a set, and relationships to other objects. Inference in KRL is called matching and is set up as a match between a pattern and an object. However, through information attached to slots, the matcher can do complex inferencing, such as following chains of implications. For example, a slot might specify a subtask that should be done in place of a direct match. The matcher makes use of an agenda (see AGENDA-BASED SYSTEMS) that can be controlled by the application to schedule these subtasks.

KL-ONE. KL-ONE (qv) is by far the most influential semantic network proposal, having spawned a number of terminological and hybrid systems (Brachman, 1979). KL-ONE frames are called concepts and the slots are called roles. Roles can have the usual attached procedures, defaults, constraints, etc. One interesting contribution is the concept of structural descriptions as a mechanism for constraining the value of more than one role in a concept. For example, one could specify that the enrollment date of a student in school precedes the graduation date with a structural description that is linked to the roles for enrollment date and graduation date and to the concept describing the precedence relationship.

KL-ONE concepts are purely descriptive. Therefore, although there is a subconcept link, there is no instance-of link. However, to distinguish concepts describing individuals rather than classes, there is an individuation link that connects individual concepts to their subsuming class concepts. For example, the individual concept (*a student whose name is John*) would be linked by an individuation link to the generic concept *Student,* which might be defined as (*persons who go to school*). Also, in order to be able to assert the existence of objects, an assertional component consisting of nexi was introduced. A nexus would be linked to all descriptions that are true of that object in the world. Nexi could be placed into various contexts, allowing several different models of the world to be explored. Finally, concepts that supply necessary and sufficient conditions on the entities described can be placed automatically into the specialization hierarchy. This was done by the KL-ONE classifier, the primary inference mechanism of the system.

KRYPTON. Krypton (Brachman and co-workers, 1983), a direct successor of KL-ONE, was one of the earliest hybrid knowledge representation systems. KRYPTON offers a clear distinction between the descriptive terminological component, called the TBox, and the assertional component, called the ABox. KL-ONE's nexi are replaced in KRYPTON by sentences in first order logic which use unary and predicates corresponding to concepts and roles defined in the TBox. The prototype implementation used a modified version of Stickel's (1982) nonclausal resolution theorem prover to implement the ABox. The KL-ONE classifier became the TBox term subsumption operation, which tests whether one concept subsumes another. Analysis of this operation has since resulted in a great deal of research on tractability vs expressiveness in the context of subsumption (see below). In addition, the KRYPTON work motivated the development of a host of hybrid knowledge representation schemes.

BACK. BACK is another hybrid representation based on KL-ONE (Nebel, 1977; Nebel and von Luck, 1988; Petalson and co-workers, 1989). As in KRYPTON, the designers of BACK emphasize the tractability of the terminological language. But in contrast to KRYPTON, BACK offers an ABox reasoner that is designed to answer queries in polynomial time. Assertions in the ABox are nondisjunctive, variable-free, and negation is not allowed. The assertional language of BACK is in fact two languages, a *Tell* language for asserting facts and an *Ask* language for querying the knowledge base. The *Tell* language allows the user to define objects using descriptions and to assert relations between them. The *Ask* language, on the other hand, is designed to correspond to the relational database query language SQL: the answers to queries are sets of tuples and set operations corresponding to the relational operators of SQL are provided. In addition, there is a formal semantics for BACK, although TBox and ABox operations are implemented with algorithms that are incomplete with respect to this semantics.

OMEGA. The OMEGA system also combines a language for forming descriptions with a logical language (Hewitt and co-workers, 1980; Attardi and Simi, 1981). This combination is done in a uniform framework that combines a calculus of descriptions with operations for forming logical statements. For example, the descriptions *charles* and ((*a man*) (*with son* = *harry*)) and the operator *is* can be related to make the assertion *charles is* ((*a man*) (*with son* = *harry*)). An interesting innovation was that logical variables could appear inside descriptions. Inference is based on a form of subsumption that takes into

account the logical statements (also called rules). There is a complete semantics for OMEGA and a complete provability relation that is known to be undecidable, so incomplete but sound inference procedures are used in the implementation. OMEGA was also interesting in that it was reflexive: it allowed statements containing descriptions of OMEGA objects to be made within the language, and a complete axiomitization of the language written in the language itself has been presented. The reflexivity also is used to specify the behavior of the inference engine in a domain-specific fashion.

CycL. A different approach to hybrid knowledge-representation is illustrated by CycL (Lenat and co-workers, 1990). CycL contains an epistemological component that uses a logic-based representation language offering the ability to refer to propositions as objects and a predicate called *ab* for representing defaults. CycL also has a heuristic level where special-purpose representations are used to support the epistemological component and where special-purpose reasoners are defined using these representations. For example, special-purpose reasoners are defined to handle transitive relations. Moreover, CycL provides a facility for automatically translating expressions (in both directions) between these two levels.

RESEARCH ISSUES

Nonmonotonicity and Defaults

When a knowledge base is incomplete, it is often necessary to make assumptions or jump to conclusions to arrive at an answer. For example, there might be an instance of the class *Bird* but what kind of bird it is might not be known. Under such circumstances, it is often reasonable to assume that because the system has not been told otherwise, this bird flies. Such assumptions have traditionally be represented in semantic networks through attributes and properties that are inherited by default and in frame-based representations through default values for slots (Fahlman, 1979). Inferences based on defaults raise some difficult problems. First, suppose that the knowledge base has been used to make inferences about some bird. If it is later discovered that this bird is in fact a penguin, any inferences arising from the assumption that it flies become invalid. This differs from knowledge bases using conventional logics where new knowledge never causes the retraction of inferences based on previously available knowledge. Schemes that have the latter property are called monotonic, whereas schemes that support any form of default reasoning are nonmonotonic (see REASONING, DEFAULT; REASONING, NONMONOTONIC). A second problem that arises is that different defaults may lead to contradictory inferences. If Nixon, for instance, has been declared to be both a Republican and a Quaker, and there are defaults that state that "unless otherwise told, a Quaker is a pacifist" but also "unless otherwise told a Republican is not a pacifist" there is clearly a problem in having the system determine Nixon's attitude toward pacifism. The closed-world assumption and frame axioms are other examples of default rules. These have been extensively used both within and outside AI, often as unstated assumptions.

There have been numerous formal proposals that attempt to deal rigorously with nonmonotonic reasoning. A pioneering proposal is Reiter's (1978) default logic, which allows inference rules of the form "if α is true and it is consistent to assume β then conclude γ." Using such rules, the default nature of the flying property of birds can be expressed with "If X is a bird and it is consistent to assume that X flies, conclude that X flies." Another pioneering approach to defaults, and perhaps the one most studied, is McCarthy's circumscription (qv) (McCarthy, 1980), which provides a mechanism for specifying schemata that state "all that is true about property P is all that is known about P." Circumscription can be used, for example, over the property of nonflying birds leading to a schema of the form "the only nonflying birds are those objects that are known to be nonflying birds." More recently, attempts have been made to formulate default reasoning in probabilistic terms (Pearl, 1989). Here "Birds fly" is interpreted as "If X is a bird, there is high probability that it flies." Such probabilistic accounts attempt to justify default inferences in terms of the (probabilistic) nature of the domain of discourse, rather than as conversational conventions.

Inconsistency

A knowledge base is inconsistent if it is possible to derive contradictory conclusions from it. As discussed above, if making default inferences, the resulting knowledge base will become inconsistent if a default is later found to be wrong. Inconsistency may also result when new knowledge simply contradicts facts already in the knowledge base. In conventional logics, inconsistency has catastrophic consequences because of so-called paradoxes of implication, which state that "anything implies a true proposition" [formally, $A \Rightarrow (B \Rightarrow A)$] and "a contradiction implies anything" (formally $A \wedge \neg A \Rightarrow B$). One approach to dealing with inconsistency is to support truth maintenance (qv) mechanisms (Doyle and London, 1980). Such mechanisms maintain a network of links representing logical relations among facts. These can be used when inconsistencies are discovered, to identify the source of these inconsistencies and to determine what facts in the knowledge base need to be revised. A second approach to inconsistency involves using logics that explicitly exclude the above paradoxes of implication. Relevance logics (Anderson and Belnap, 1975) achieve this by narrowing the notion of proof: A entails B in relevance logics if there is a proof of B that actually uses A during one of its steps. Relevance logics have turned out to be interesting both from a computational point of view (Patel-Schneider, 1985) and as a useful vehicle for formal models of knowledge and belief (Levesque, 1984).

Incompleteness

Except for toy situations, knowledge bases have incomplete information about their universe of discourse. This incompleteness may arise due to insufficient information about that universe (consider a knowledge base that has

not yet been told anything, for instance). It may also come about because of a weak provability relation. For instance, a logic-based knowledge representation system with no inference rules will not be able to infer $\alpha \vee \beta$ from a knowledge base containing both α and β. Alternatively, incompleteness may arise because the inference procedure (implementation of the provability relation) is incomplete, for computational economy, say, or even because the logic is powerful enough so that Godel's incompleteness theorem holds.

From an expressiveness point of view, incompleteness poses serious problems. The facts below, for instance, contain incomplete information.

1. "George or Mary graduated from the University of Toronto."
2. "There are 40,000 registered students at the university."
3. "The morning star is the same as the evening star."

For example, sentence 1 does not specifically relate the state of affairs, sentence 2 declares that all members of a class have a property without specifying who the members are, and sentence 3 declares that two nonidentical expressions refer to the same individual. Moreover, there may be a need to express facts about the completeness or incompleteness of the knowledge base itself, such as the closed-world assumption, used by databases (see above), or

4. "There is an unknown student."

Logic-based schemes that subsume first-order logic can handle sentences 1–3 in a satisfactory way using logical connectives and quantifiers. Expressing these facts in procedural and semantic network schemes is more problematic. Sentence 4, however, is a problem even in first-order logic and has been studied at length in the literature. For example, Levesque (1984) introduces a new operator K that when placed in front of a fact is read as "it is known that . . ." It is then possible to state that there is an unknown student with

$$(\exists X)[\text{student}\ (X) \wedge \neg K\text{student}\ (X)]$$

that is, there is a student who is not known to be so. On the other hand, there have been attempts to deal with incompleteness within the framework of first-order logic (Moore, 1980) by introducing a predicate *Know,* which takes as arguments an agent (the "knower") and an encoded formula (the known fact). In both proposals possible world semantics serve as a basis for a semantic theory.

Expressiveness vs Tractability

Expressiveness vs. tractability trade-offs are a fact of life in computer science and have been known at least since the development of formal language and automata theory (Hopcroft and Ullman, 1969). In the context of knowledge representation, Brachman and Levesque (1985) are credited with the introduction of this issue as well as first results focusing on the terminological language FL, a cleaned-up subset of KL-ONE. FL includes, among others, a term-forming operator *RESTR* that allows type constraints on attributes, as in

(AND person (ALL (RESTR friend male))

which designates the class "persons all of whose friends are male." Brachman and Levesque show that subsumption in a language without *RESTR* can be determined by an algorithm of complexity $O(n^2)$, where n is the size of the operands, while the same operation in FL is as hard as satisfiability for propositional calculus and, therefore, intractable. This surprising result has led to a broad study of trade-off issues for terminological languages, with more noteworthy results. For example Schmidt-Schauss (1989) proves that subsumption in KL-ONE is, in fact, undecidable, while (Patel-Schneider, 1986) establishes that subsumption can be made tractable even for expressively rich terminological languages if the semantics of the language is weakened.

More recently, complexity results have been reported for many knowledge representation problems. (Borgida and Etherington, 1989) proves, for example, that fast approximate algorithms exist for query evaluations in knowledge bases that include ground clauses, generalization hierarchies, and disjunctive formulas, whereas exact query evaluation algorithms are expensive due to the presence of disjunctive information. Kautz and Selman (1989) studied the complexity of problems such as finding a maximal consistent set of conclusions given a propositional default theory and showed that this problem is tractable only in special cases. Their results suggest that it is easier to fill-out (vivify) an incomplete knowledge base using default rules than to use these rules as extension to normal resolution-type theorem proving. Villain and Kautz (1986) and Villain, Kautz, and van Beek (1989) summarize complexity results on inference with respect to a temporal logic based on intervals, originally proposed by Allen (1981). The decision problem with respect to the original temporal logic is shown to be intractable, but several examples of weaker logics are proven to have tractable decision problems.

Ontologies

The nature of the domain of discourse plays an important role in determining the features of a knowledge representation scheme. For example, if the domain involves time, the knowledge representation scheme may offer appropriate constructs, such as time points or intervals along with temporal relations, for representing temporal facts such as "Mary married before graduating from university." In addition, a specialized inference engine may be used for reasoning with respect to temporal knowledge, using results such as those reported by Allen (1981).

Work on ontologies involves formalizing the properties of a particular class of objects that are supposed to populate the universe of discourse, be they time points or spatial segments, agents performing actions and having goals or knowledge items. Here is a quick overview of some ontologies studied in the literature.

Entities and Relationships. Most representation schemes assume that the world is populated by entities that are endowed with a unique and immutable identity, a lifetime, and relationships to other entities. Basic as this ontology may seem, it is by no means universal. For instance, Hayes (1985) offers an ontology for material substances where entities (say, a liter of water and a pound of sugar) can be merged resulting in a different entity.

Causality and Time. As indicated earlier, time can be modeled in terms of points or intervals, ordered linearly or partially (Allen, 1984). Causality imposes existence constraints on events. If event A causes event B and A has been observed, B can be expected as well, possibly with some time delay. Formal models of causality have been published (McCarthy, 1968; Rieger, 1976).

Space. Space is thought of in terms of two- or three-dimensional points or larger units, including spheres, cubes, pyramids, etc (eg, Davis, 1986).

Agents and Actions. The domain of discourse includes here agents having beliefs and goals and being capable of carrying out actions (Allen, 1981). Maida and Shapiro (1982), among others, address the problem of representing propositional attitudes, such as beliefs, desires, and intentions for agents.

Existence. A distinction is made here between different modes of existence, including physical existence, such as that of the editor of this volume; abstract existence, such as that of the number 7; nonexistence, characteristic of Santa Claus or a canceled trip to Japan; and impossible existence, such as that of the square root of -1 or the proverbial square circle (Hirst, 1989). General ontologies have been studied in philosophy (Carnap, 1967; Bunge, 1977) and may prove useful starting points for future work on ontologies.

Knowledge Base Management

The practice of building and using large knowledge bases imposes a number of requirements on knowledge representation systems. In particular, experience in developing, deploying, and maintaining knowledge bases suggests a number of management facilities that may be supported to a greater or lesser extent by a knowledge representation system. Explanation facilities, available since the very first expert systems (Shortliffe, 1976), allow the system to explain its line of reasoning through *why* and *how* questions, thereby helping the user accept it or determine where it is faulty. Knowledge acquisition has been recognized as a bottleneck in building knowledge bases. State-of-the-art acquisition facilities support a number of mechanisms for acquiring knowledge, including learning of rules from raw data (Boose, 1989) and extraction of knowledge from linguistic expressions. Truth maintenance facilities allow the system to maintain information on logical interdependencies among facts (such as fact α was derived from facts β and γ), which can then be used to restore a knowledge base (that contains nonmonotonic inferences) to consistency after updates (Doyle, 1979). Validation facilities help the user validate the contents of the knowledge base in a systematic fashion (Ginsberg, 1988). Documentation facilities document the contents of the knowledge base, using natural language and recent technologies such as hypertext (Jansen and Compton, 1988). Maintenance facilities include ones for knowledge base evolution, forgetting, summarizing, and related functions humans find useful in managing a body of knowledge.

In addition, effective management of large knowledge bases presupposes some of the functionality of data-base management systems, including the following. *Persistence* is the ability of a knowledge base to persist beyond the execution of the processes that create and access it. Persistence mechanisms for databases are supposed to be both local (ie, can be associated with a particular datum independently of its type) and implicit (in the sense that the user does not need to specify explicitly that data are to persist). *Secondary storage management* is another standard management feature for databases. Database management systems offer a variety of techniques, such as index management, data clustering, data buffering, access path selection, and query optimization to support the efficient use of secondary storage. *Concurrency and recovery* are taken for granted for database management systems and allow multiple users to efficiently share a database. Mylopoulos and Brodie (1986) and Schmidt and Thanos (1989) present collections of ideas about knowledge base management and the research issues it raises.

CONCLUSIONS

Controversies about what knowledge representation is or does notwithstanding, many schemes have been proposed as general frameworks for knowledge representation and reasoning. These have, in turn, led to hundreds of commercial products and thousands of industrially deployed knowledge bases, mostly as components of expert systems. Despite these successes, many research issues remain unresolved, including ontological and semantic ones for knowledge representation schemes as well as implementation-oriented ones for knowledge representation systems. Apart from expert system applications, knowledge representation is increasingly influential in other areas of computer science, such as databases (Mylopoulos and Brodie, 1988) and software engineering (Rich and Waters, 1986), where "capturing knowledge about the world" is becoming an integral part of the process of building software systems.

BIBLIOGRAPHY

H. Ait-Kaci, *A New Model of Computation Based on a Calculus of Type Subsumption,* MS-CIS-83-40, Department of Computer and Information Science, The Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, 1983.

J. F. Allen, *A General Model of Action and Time,* Technical Re-

port, Department of Computer Science, University of Rochester, N.Y., 1981.

J. F. Allen, "Towards a General Theory of Action and Time," *Artif. Intell.* **23,** 123–154 (1984).

J. Anderson, *Cognitive Psychology and its Implications,* W. H. Freeman, New York, 1990.

A. R. Anderson and N. D. Belnap, *Entailment: The Logic of Relevance and Necessity,* Vol. I, Princeton University Press, Princeton, N.J., 1975.

G. Attardi and M. Simi, *Semantics of Inheritance and Attributions in the Description System Omega.* Technical Report **S-81-16,** Universita di Pisa, Pisa, Italy, 1981.

D. G. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language," *Cogn. Sci.* **1,** 3–46 (1977).

J. Boose and B. Gaines, eds., *Knowledge Acquisition Tools for Expert Systems,* Academic Press, Inc., Orlando, Fla., 1989.

A. Borgida and D. W. Etherington, "Hierarchical Knowledge Bases and Efficient Disjunctive Reasoning," in *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, 1989, pp. 33–43.

R. J. Brachman, "On the Epistemological Status of Semantic Networks," in Findler, 1979.

R. J. Brachman, R. E. Fikes, and H. J. Levesque, "Krypton: A Functional Approach to Knowledge Representation," *IEEE Comp.* **16**(10), 67–74 (Oct. 1983).

R. J. Brachman and H. J. Levesque, "A Fundamental Tradeoff in Knowledge Representation and Reasoning" in R. J. Brachman and H. J. Levesque, eds., *Readings in Knowledge Representation,* Morgan-Kaufmann, San Mateo, Calif., 1985, pp. 41–70.

M. Bunge, *Treatise on Basic Philosophy: Ontology I—The Furniture of the World,* Reidel, 1977.

R. Carnap, *The Logical Structure of the World: Pseudoproblems in Philosophy,* University of California Press, 1967.

N. Cercone and G. McCalla, eds., *The Knowledge Frontier,* Springer-Verlag, New York, 1987.

A. G. Cohn, "On the Appearance of Sortal Literals: a Non Substitutional Framework for Hybrid Reasoning," in Borgida and Etherington, pp. 55–66.

C. Date, *An Introduction to Database Systems,* Vols I–II, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1975.

E. Davis, *Representing and Acquiring Geographic Knowledge,* Pitman, New York, 1986.

R. Davis, "Meta-rules: Reasoning about Control," *Artif. Intell* **15**(3), 179–222 (1980).

J. Doyle, "A Glimpse of Truth Maintenance," in P. Winston and R. Brown, eds., *Artificial Intelligence: An MIT Perspective,* MIT Press, Cambridge, Mass., 1979, pp. 119–136.

J. Doyle and P. London, "A Selected Descriptor-Indexed Bibliography to the Literature of Belief Revision," *SIGART Newslett.* **71,** 7–23 (Apr. 1980).

L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The HEARSAY-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Comput. Surv.* **12**(2), 213–253 (June 1980).

S. E. Fahlman, *NETL: A System For Representing and Using Real World Knowledge,* MIT Press, Cambridge, Mass., 1979.

N. V. Findler, ed., *Associative Networks: Representation and Use of Knowledge by Computers,* Academic Press, Inc., New York, 1979.

A. M. Frisch and J. F. Allen, "Knowledge Retrieval as Limited Inference," in *Proceedings, Sixth Annual international Conference on Automated Deduction,* D. W. Loveland, ed., Springer-Verlag, New York, 1982.

B. V. Funt, "Problem-Solving with Diagrammatic Representations" *Artif. Intell.* **13**(3), 201–230 (1980).

M. R. Genesereth, "An Overview of Metalevel Architecture," in *Proceedings of the Third National Conference on Artificial Intelligence,* Washington, D.C., AAAI, Menlo Park, Calif., 1983.

A. Ginsberg, "Knowledge Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy," in *Proceedings of the Seventh National Conference on Artificial Intelligence,* St. Paul, Minn., AAAI, Menlo Park, Calif., 1988, pp. 585–589.

I. P. Goldstein and R. B. Roberts, "Nudge: a Knowledge-Based Scheduling Program," in *Proceedings of the Fifth IJCAI,* Cambridge, Mass., Morgan-Kaufmann, San Mateo, Calif., 1977, pp. 257–263.

P. J. Hayes, "In Defense of Logic," in Goldstein and Roberts, 1977, pp. 559–65.

P. J. Hayes, "The Second Naive Physics Manifesto," in J. R. Hobbs and R. C. Moore, eds., *Formal Theories of the Commonsense World,* Ablex Publishing Corp., Norwood, N.J., 1985, pp. 1–36.

F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, eds., *Building Expert Systems,* Addison-Wesley Publishing Co., Inc., Reading, Mass., 1983.

G. G. Hendrix, "Encoding Knowledge in Partitioned Networks," in N. V. Findler, ed., *Associative Networks: Representation and Use of Knowledge by Computers,* Academic Press, Inc., New York, 1979, pp. 51–92.

C. Hewitt, "PLANNER: A Language for Proving Theorems in Robots," in *Proceedings of the Second IJCAI,* London, Morgan-Kaufmann, San Mateo, Calif., 1971.

C. Hewitt, G. Attardi, and M. Simi, "Knowledge Embedding in the Description System Omega," in *Proceedings of the First National Conference on Artificial Intelligence,* Stanford, Calif., 1980, pp. 157–164.

G. Hirst, "Ontological Assumptions in Knowledge Representation," in Borgida and Etherington, 1989, pp. 157–169.

J. E. Hopcroft and J. D. Ullman, *Formal Languages and their Relation to Automata,* Addison-Wesley Publishing Co., Inc., Reading, Mass., 1969.

G. Hulin, A. Pirotte, D. Roelants, and M. Vauclair, "Logic and Databases," in A. Thayse, ed., *From Modal Logic to Deductive Databases–Introducing a Logic-Based Approach to Artificial Intelligence,* John Wiley & Sons, Inc., 1989.

R. Jansen and P. Compton, "The Knowledge Dictionary: An Application of Software Engineering Techniques to the Design and Maintenance of Expert Systems," in *Proceedings of the Seventh National Conference on Artificial Intelligence,* St. Paul, Minn., AAAI, Menlo Park, Calif., 1988.

H. A. Kautz and B. Selman, "Hard Problems for Simple Default Logics," in Borgida and Etherington, 1989, pp. 189–197.

S. E. Keene, *Object-Oriented Programming in COMMON LISP: A Programmer's Guide to CLOS,* Addison-Wesley, 1989.

R. Kowalski, *Logic for Problem Solving,* Elsevier North Holland, New York, 1979.

B. M. Kramer, "Control of Reasoning in Knowledge-Based Systems," Ph.D. Thesis, University of Toronto, Toronto, Ontario, 1986.

D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "CYC: Towards Programs with Common Sense," *Communications of the ACM,* **33,** pp. 30–49, 1990.

H. J. Levesque, "A Fundamental Tradeoff in Knowledge Representation and Reasoning," in *Canadian Society for Computational Studies of Intelligence/Société Canadienne pour l'Étude de l'Intelligence par Ordinateur, Proceedings of the Fifth Biennial Conference,* London, Ont., Canada, 1984, pp. 141–152.

H. J. Levesque, "A Logic of Implicit and Explicit Belief," in *Proceedings of the Fourth National Conference on Artificial Intelligence,* Austin, Tex., AAAI, Menlo Park, Calif., 1984, pp. 198–202.

H. J. Levesque, "Knowledge Representation and Reasoning, *Ann. Rev. Comput. Sci.,* no. 1, 255–287 (1986).

H. J. Levesque and J. Mylopoulos, "A Procedural Semantics for Semantic Networks," in Findler, 1979, pp. 93–120.

J. McCarthy, "Programs with Common Sense," in Minsky, 1968, pp. 403–418.

J. McCarthy, "Circumscription—A Form of Non-monotonic Reasoning," *Artif. Intell.* **13,** 27–39 (1980).

J. McDermott and C. Forgy, "Production System Conflict Resolution Strategies," in Waterman and Hayes-Roth, eds., 1978, pp. 177–199.

A. Maida and S. C. Shapiro, "Intensional Concepts in Propositional Semantic Networks," *Cogn. Sci.* **6,** 291–330 (1982).

M. Minsky, ed., *Semantic Information Processing,* MIT Press, Cambridge, Mass., 1968.

M. Minsky, "A Framework for Representing Knowledge," in P. H. Winston, ed., *The Psychology of Computer Vision,* McGraw-Hill Book Co., Inc., New York, 1975, pp. 211–277.

R. C. Moore, *Reasoning about Knowledge and Action,* Technical Note **284,** AI Centre, SRI International, Palo Alto, Calif., 1980.

J. Mylopoulos and M. Brodie, eds., *On Knowledge Base Management Systems: Integrating AI and Database Technologies,* Springer-Verlag, New York, 1986.

J. Mylopoulos and M. Brodie, eds., *Readings in AI and Databases,* Morgan-Kaufmann, San Mateo, Calif., 1988.

B. Nebel, "Computational Complexity of Terminological Reasoning in BACK," *Artif. Intell.* **34,** 371–383 (1988).

B. Nebel and K. von Luck, "Hybrid Reasoning in BACK," in Z. W. Ras and L. Saitta, eds., *Methodologies for Intelligent Systems,* Vol. 3, North-Holland, Amsterdam, The Netherlands, 1988.

A. Newell, "Production Systems: Models of Control Structure," in W. Chase, ed., *Visual Information Processing,* Academic Press, Inc., New York, 1973.

A. Newell and H. A. Simon, *Human Problem Solving,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972.

P. F. Patel-Schneider, "A Decidable First Order Logic for Knowledge Representation," in *Proceedings of the Ninth IJCAI,* Los Angeles, Calif., Morgan-Kaufmann, San Mateo, Calif., 1985, pp. 455–458.

P. F. Patel-Schneider, "A Four-Valued Semantics for Frame-Based Description Languages," in *Proceedings of the Fifth National Conference on Artificial Intelligence,* Philadelphia, AAAI, Menlo Park, Calif., 1986, pp. 344–348.

J. Pearl, "Probabilistic Semantics for Nonmonotonic Reasoning: A Survey," in Borgida and Etherington, 1989, pp. 505–526.

C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz, *The BACK System Revisited,* KIT Report **75,** Technical University of Berlin, 1989.

R. Reiter, "On Closed World Data Bases," in H. Gallaire and J. Minker, eds., *Logic and Databases,* Plenum, New York, 1978, pp. 55–76.

R. Reiter, "A Logical Reconstruction of Relational Theory," in M. Brodie, J. Mylopoulos, and J. Schmidt, eds., *On Conceptual Modelling: Perspectives from AI, Databases and Programming Languages,* Springer-Verlag, New York, 1984.

C. Rich and R. C. Waters, eds., *Readings in Artificial Intelligence and Software Engineering,* Morgan-Kaufmann, San Mateo, Calif. 1986.

C. Rieger, "An Organization of Knowledge for Problem-Solving and Language Comprehension," *Artif. Intell.* **7**(2), 89–127 (1976).

J. A. Robinson, *Logic: Form and Function,* Edinburgh University Press, Edinburgh, UK, 1979.

J. Schmidt and C. Thanos, eds., *Foundations of Knowledge Base Management,* Springer-Verlag, New York, 1989.

M. Schmidt-Schauss, "Subsumption in KL-ONE is Undecidable," in Brogida and Etherington, 1989, pp. 505–526.

L. K. Schubert, "Extending the Expressive Power of Semantic Networks," *Artif. Intell.* **7,** 163–198 (1976).

S. C. Shapiro, "A Net Structure for Semantic Storage, Deduction, and Retrieval," in Hewitt, 1971.

S. C. Shapiro, "The SNePS Semantic Network Processing System," in Findler, 1979.

E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN,* Elsevier Science Publishing Co., Inc., New York, 1976.

J. Sowa, ed., *Principles of Semantic Networks,* Morgan-Kaufmann, San Mateo, Calif., 1991.

M. E. Stickel, "A Non-Clausal Connection Graph Resolution Theorem Proving System," in *Proceedings of the Second National Conference on Artificial Intelligence,* Pittsburgh, Pa., AAAI, Menlo Park, Calif., 1982, pp. 229–233.

G. J. Sussman and D. V. McDermott, *Why Conniving Is Better Than Planning,* Technical Report MIT-AI-225A, Cambridge, Mass., 1972.

M. Vilain and H. Kautz, "Constraint Propagation Algorithms for Temporal Reasoning," in Patel-Schneider, 1986, pp. 377–382.

D. A. Waterman and F. Hayes-Roth, eds., *Pattern-Directed Inference Systems,* Academic Press., Inc., New York, 1978.

W. A. Woods, "What's in a Link: Foundations for Semantic Networks," in D. G. Bobrow and A. Collins, eds., *Representation and Understanding,* Academic Press, Inc., New York, 1975, pp. 35–82.

General References

J. Allen and P. Hayes, "A Common Sense Theory of Time," in *Proceedings of the Ninth IJCAI,* Los Angeles, Calif., Morgan-Kaufmann, San Mateo, Calif., 1985.

A. Barr and J. Davidson, "Representation of Knowledge," in A. Barr and E. A. Feigenbaum, eds., *The Handbook of Artificial Intelligence,* Vol. 1, W. Kaufmann, Los Altos, Calif., 1981, pp. 140–121.

R. J. Brachman and H. J. Levesque, eds., *Readings in Knowledge Representation,* Morgan-Kaufmann, San Mateo, Calif., 1985.

R. J. Brachman and B. C. Smith, eds., *Special Issue on Knowledge Representation, SIGART Not.* 70 (1980).

A. M. Frisch and J. F. Allen, "Knowledge Retrieval as Limited Inference," in D. W. Loveland, ed., *Proceedings of the Sixth International Conference on Automated Deduction,* Springer-Verlag, New York, 1982.

G. Hulin, A. Pirotte, D. Roelants, M. Vauclair, "Logic and Databases," in A. Thayse, ed., *From Modal Logic to Deductive Databases—Introducing a Logic-Based Approach to Artificial Intelligence,* John Wiley & Sons, Inc., New York, 1989.

G. McCalla and N. Cercone, eds., *Special Issue on Knowledge Representation, IEEE Comput.* **16**(10) (Oct. 1983).

W. A. Martin, "Descriptions and the Specializations of a Concept," in P. Winston and R. Brown, eds., *Artificial Intelligence: An MIT Perspective,* MIT Press, Cambridge, Mass., 1979, pp. 379–419.

J. Mylopoulos and H. J. Levesque, "An Overview of Knowledge Representations," in M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, eds., *On Conceptual Modelling,* Springer-Verlag, New York, 1984, pp. 3–18.

M. R. Quillian, "Semantic Memory," in M. Minsky, ed., *Semantic Information Processing,* MIT Press, Cambridge, Mass., 1968, pp. 227–270.

M. Vilain, H. Kautz, and P. van Beek, "Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report," in D. S. Weld and J. de Kleer, eds., *Readings in Qualitative Reasoning about Physical Systems,* Morgan-Kaufmann, San Mateo, Calif., 1989.

BRYAN KRAMER
JOHN MYLOPOULOS
University of Toronto

KOKON

KOKON is a system that generates contracts for selling real estate. It was developed in 1988 in collaboration at Technical University Munich and Systemtechnik Berner & Mattner GmbH under the sponsorship of the German government. KOKON integrates techniques of legal reasoning and belief revision (qv). (See D. Kowalewski, J. Schneeberger, and S. Wiefel, "KOKON-3: Ein prototypisches System zur wissensbasierten Vertragskonfigurierung," in M. Paul, ed., *GI-19. Jahrestagung*, Springer, 1989, pp. 79–92.)

J. SCHNEEBERGER
TH Darmstadt

KRL

A frame-based language for knowledge representation (qv), KRL was developed by Bobrow and Winograd in 1977 (see D. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language," *Cog. Sci.* **1**, 3–46 (1977); T. Winograd, "Frame Representations and the Declarative/Procedural Controversy," in D. Bobrow and A. Collins, eds., *Representation and Understanding: Studies in Cognitive Science*, Academic Press, New York).

T. WINOGRAD
Stanford University

ENCYCLOPEDIA OF ARTIFICIAL INTELLIGENCE

This extensively revised and expanded *Second Edition* of the *Encyclopedia of Artificial Intelligence* defines the discipline by bringing together the core of knowledge from all fields encompassed by AI. It covers the latest developments in current AI topics such as neural networks, fuzzy logic, machine vision, natural language generation, and many more. Includes:

- Over 450 articles—all entries written expressly for the *Encyclopedia*
- Over 5,000 literature references; 454 illustrations and color photographs
- Over 50% new and revised material
- Exemplary indexing and cross-referencing for easy, complete information access to all topics

Praise for the *First Edition*...

"The *Encyclopedia* is a wonder of clarity and scope: surprisingly easy to read...the clarity is an especially pleasant surprise, considering the articles were all written by AI experts...It's a treasure house of easily accessible knowledge."
—*Language Technology*

"Excellent bibliographies are attached to most of the articles, and diagrams and sketches are clear and helpful. The indexing and cross-indexing are exemplary. As the editor points out, the reader will be led by the extensive cross-references to almost every other article..." —*Artificial Intelligence Reporter*

"The *Encyclopedia* is a first-class piece of work that will be an indispensable part of any AI library." —*Computing Reviews*

"...A tour de force...a truly fantastic encyclopedia which no one in the field of artificial intelligence can afford to be without."
—*Systems Research & Information*

WILEY-INTERSCIENCE
John Wiley & Sons, Inc.
Professional, Reference and Trade Group
605 Third Avenue, New York, N.Y. 10158-0012
New York • Chichester • Brisbane • Toronto • Singapore
ISBN 0 471-50307-X (Two-Volume Set)
ISBN 0 471-50305-3 (Vol. 1)
ISBN 0 471-50306-1 (Vol. 2)

ISBN 0-471-50305-3

9 780471 503057 90000