Fads and Fallacies about Logic

John F. Sowa

VivoMind Intelligence, Inc.

Throughout the history of AI, logic has been praised by its admirers, maligned by its detractors, and discussed in confusing and misleading terms by almost everybody. Among the pioneers of AI, John McCarthy has always been a strong promoter of logic, but Marvin Minsky has been a skeptic who experimented with a wide range of alternatives. Roger Schank had no doubts about logic, which he denounced at every opportunity. He introduced the distinction between the *neats* who used logic for everything vs. the *scruffies* like himself who developed notations that were specifically designed for the problem at hand. Even advocates of logic have disagreed among themselves about the role of logic, the subset appropriate to any particular problem, and the tradeoffs of ease of use, expressive power, and computational complexity. The debates introduced many valuable ideas, but the hype and polemics confused the issues and often led to unfortunate design decisions. Controversies arise in several areas: relationships between language and logic; the range of notations for logic; procedural vs. declarative representations; distinctions between object level and metalevel statements; expressive power and complexity; and the role of logic in readable, usable, efficient interfaces.

This is a preprint of an article that appeared in *IEEE Intelligent Systems*, March 2007, **22:2**, pp. 84-87.

1. Language and Logic

No discussions about logic have been more confused and confusing than the debates about how logic is related to natural languages. Historically, logic evolved from language. Its name comes from the Greek *logos*, which means word or reason and includes any language or method of reasoning used in any of the *-ology* fields of science and engineering. Aristotle developed formal logic as a systematized method for reasoning about the meanings expressed in ordinary language. For the next two millennia, formal logic was expressed in a stylized or *controlled* subset of a natural language: originally Greek, then Latin, and later modern languages.

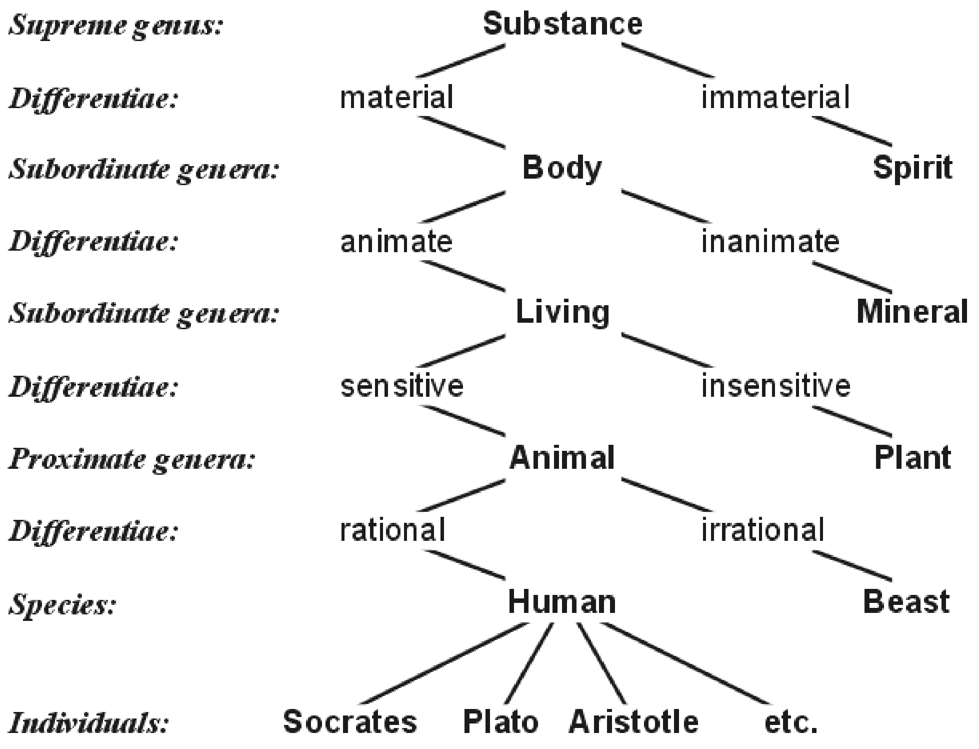
In the 19th and 20th centuries, mathematicians took over the development of logic in notations that diverged very far from its roots, but every operator of any version of logic is a specialization of some word or phrase in natural language: ∃ for *there exists*, ∀ for *every*, ∧ for *and*, ∨ for *or*, ⊃ for *if-then*, ~ for *not*, ◊ for *possibly*, and □ for *necessarily*. The metalevel words for talking about logic and deduction are the same words used for the corresponding concepts in natural languages: *truth, falsity, reasoning, assumption, conclusion,* and *proof.* Although mathematical logic may look very different from ordinary language, every formula in any notation for logic can always be translated to a sentence that has the same meaning. Furthermore, every step of every proof in any formal logic can be translated to an argument in ordinary language that is just as correct and cogent as the formal version.

What makes formal logic hard to use is its rigidity and its limited set of operators. Natural languages are richer, more expressive, and much more flexible. That flexibility permits vagueness, which some logicians consider a serious flaw, but a precise statement on any topic is impossible until all the details are determined. As a result, formal logic can only express the final result of a lengthy process of analysis and design. Natural language, however, can express every step from the earliest hunch or tentative suggestion to the finished specification.

In short, there are two equal and opposite fallacies about language and logic: at one extreme, logic is considered unnatural and irrelevant; at the opposite extreme, language is incurably vague and should be replaced by logic. A more balanced view must recognize the virtues of both: logic is the basis for precise reasoning in every natural language; but without vagueness in the early stages of a project, it would be impossible to explore all the design options.

2. What is Logic?

Unreadability is a common complaint about logic, but that is only true of 20th century mathematical logic. In the middle ages, the usual notation was a controlled natural language, but diagrams were used as a supplement. The *Tree of Porphyry* displayed Aristotle's categories in a type hierarchy, and Ramon Lull invented a method of defining categories by rotating and aligning disks with inscribed attributes. Inspired by Lull's system, Leibniz used algebra to define categories by conjunctions of attributes, which he encoded as prime numbers. In the 19th century, Boole used algebra to represent propositions, and Frege invented a tree notation to represent the quantifiers and operators of first-order logic. Peirce represented FOL by adding quantifiers to Boolean algebra, but he later invented *existential graphs* as a logically equivalent notation.



Tree of Porphyry as drawn by Peter of Spain (1239)

To accommodate all the variations, a logic may be defined as any precise notation for expressing statements that can be judged true or false. A *rule of inference* is a truth-preserving transformation: when applied to a true statement, the result is guaranteed to be true. To clarify the notion of "judging," Tarski defined a *model* as a set of entities and a set of relationships among those entities. *Model theory* is a systematic method for evaluating the truth of a statement in terms of a model. According to these definitions, a database, in relational or network format, can be considered a model, and the method of evaluating the WHERE clause of an SQL query is equivalent to Tarski's evaluation function.

This definition is broader than usual. Many logicians limit logic to a narrow range of mathematical notations, and most nonlogicians do not think of their notations as a kind of logic. Yet almost any declarative notation, graphic or linear, could be treated as a version of logic: just specify it precisely, and define an evaluation function or a translation to some logic that already has an evaluation function.

3. Deriving Procedures from Declarations

A procedure is used in only one way, but a declarative specification can be used in many different ways. Directions from a highway to a hotel, for example, specify a procedure for following a single path, but a map is a declarative specification that determines all possible paths. With a map, a person or computer can derive a procedure by tracing a path between any two points. In computer science, grammars are logic-based declarations that can be used to analyze a sentence, generate a sentence, detect errors in a sentence, or suggest corrections to a sentence.

For over forty years, debates about procedural or declarative languages are periodically revived. In database systems, tables and graphs are equivalent ways of expressing data, but queries about the data can be represented in two ways: a path-based procedure for navigating the data; or a declarative statement in a logic-based language such as SQL. In the 1970s, the debate was settled in favor of SQL, but object-oriented databases have revived the argument for a procedural approach:

- Data represented in a tree or graph can be viewed as a road map.
- Procedural directions are the most efficient way to tell a computer how to get from one point to another.
- If the programmer specifies the access path, the computer does not need complex algorithms to derive the path.

For some systems, these arguments are valid. But the following arguments killed the path-based CODASYL DBTG query language in the 1970s:

- Tables and graphs specify logically equivalent access paths.
- Optimizing algorithms can derive more efficient paths than most programmers.
- A query that takes a few lines to declare in SQL may expand to several pages of highly error-prone procedural directions.

These arguments are just as valid today. The major vendors of object-oriented databases support SQL as an alternative to the path-based procedures because users demand it. When given the choice, they prefer SQL, even for databases whose native organization is a network.

In summary, the choice of procedural or declarative methods depends on the available tools. Since current computers execute procedures, declarations must be translated to procedures. Although declarations allow different procedures to be generated for different purposes, good performance requires good optimization algorithms. If no optimizer is available, a procedural language that can be translated directly to machine language may be necessary.

4. Object Language and Metalanguage

Language about language or *metalanguage* is ubiquitous. As an example, the following sentence was spoken by a child named Laura at age 34 months and recorded by Limber (1973):

When I was a little girl, I could go "Geek geek," like that;
but now I can go "This is a chair."

This sentence involves metalanguage about the quotations. But logically, the modal auxiliaries *can* and *could* may be represented as metalanguage about the containing clauses (Sowa 2003). As a result, Laura's sentence can be represented by two levels of metalogic about the quotations.

In adult speech, metalanguage is so common that it's often unrecognized. The modal verbs, which can also be expressed by adverbs such as *possibly*, are a special case of expressions such as *probably*, *very likely*, *I doubt that...*, or *The odds are 50 to 1 that....* All versions of modality, probability, certainty, or fuzziness can be represented by first-order statements about first-order statements. Contrary to claims that logic can't handle uncertainty, the metalevel predicates in such sentences can be defined by first-order axioms stated at the next higher metalevel.

Some people argue that metalevel representations are complex and inefficient. But for many applications, metalanguage can significantly reduce the complexity, as in the following sentences in controlled English and their translations to an algebraic notation:

"Every dog is an animal" $\Rightarrow$ $(\forall x)(\mathrm{dog}(x) \supset \mathrm{animal}(x))$.

"Dog is a subtype of Animal" $\Rightarrow$ $\mathrm{Dog} < \mathrm{Animal}$.

The first sentence is translated to predicate logic, but the second is a metalevel statement about types, which can be represented with the relation < between two constants. Such statements define a type hierarchy, which can be encoded in bit strings or by products of primes. Subsequent reasoning with a *typed* or *sorted* logic can replace a chain of inferences with a divide instruction or a bit-string comparison.

These examples illustrate an important use of metalanguage: stratify the knowledge representation in levels that can be defined separately or be processed by simpler algorithms. The type hierarchy, in particular, is an important level that can be defined by Aristotle's syllogisms or *description logics*. It is usually considered a privileged level with greater *entrenchment* than ordinary assertions. That privilege gives it a modal effect of being *necessarily true*. As another example of entrenchment, database constraints are obligatory with respect to ordinary assertions called *updates*. Three levels — a type hierarchy, database constraints, and updates — can support *multimodal reasoning* with a mixture of modes that are necessary or obligatory for different reasons.

5. Expressive Power and Computational Complexity

Computational complexity is a property of an algorithm. Since a program implements an algorithm, the complexity of the program is determined by the complexity of the algorithm it embodies. As a declarative language, logic may be used for different purposes, which may be supported by programs with different levels of complexity. In first-order logic, for example, a proof may take an exponential amount of time or even cause a theorem prover to loop forever. Yet the worst cases rarely occur, and theorem provers can be efficient on the FOL statements people actually use. When Whitehead and Russell wrote the *Principia Mathematica*, theories of computational complexity were unknown. Yet when applied to their theorems in propositional and first-order logic, the program by Wang (1960) proved all 378 in just seven minutes. That was an average of 1.1 seconds per theorem on an IBM 704, a vacuum-tube machine with an 83-kilohertz CPU and 144K bytes of storage.

For database queries and constraints, SQL supports full FOL, but even the worst case examples can be evaluated in polynomial time, and the most common queries are evaluated in linear or logarithmic time. That performance enables SQL algorithms to process terabytes or petabytes of data and makes first-order logic, as expressed in SQL, the most widely used version of logic in the world. As an example, the SQL translation of the following query would be answered in logarithmic time if the employee relation is indexed on the name field:

Find John Doe's department, manager, and salary.

If there are N employees, the time for the following query is proportional to (N log N):

Find all employees who earn more than their managers.

This query would take N steps to find the manager and salary of each employee. The (log N) factor is needed to find the salary of each employee's manager.

Even complex queries can be evaluated efficiently if they process a subset of the database. Suppose that the complex condition in the following query takes time proportional to the cube of the number of entries:

For all employees in department C99, find [*complex condition*].

If a company has 10,000 employees, N^3 would be a trillion. But if there are only 20 employees in department C99, then 20^3 is only 8,000.

In summary, computational complexity is important. But complexity is a property of algorithms, and only indirectly a property of problems, since most problems can be solved by different algorithms with different complexity. The language in which a problem is stated has no effect on complexity. Reducing the expressive power of a logic does not solve any problems faster; its only effect is to make some problems impossible to state.

6. Using Logic in Practical Systems

The hardest task of knowledge representation is to analyze knowledge about a domain and state it precisely in any language. Since the 1970s, knowledge engineers and systems analysts have been eliciting knowledge from domain experts and encoding it in computable forms. Unfortunately, the tools for database design have been disjoint from expert-system tools; they use different notations that require different skills and often different specialists. If all the tools were based on a common, readable notation for logic, the number of specialists required and the amount of training they need could be reduced. Furthermore, the domain experts would be able to read the knowledge representation, detect errors, and even correct them.

The first step is to support the notations that people have used for logic since the middle ages: controlled natural languages supplemented with type hierarchies and related diagrams. Although full natural language with all its richness, flexibility, and vagueness is still a major research area, the technology for supporting controlled NLs has been available since the 1970s. Two major obstacles have prevented such languages from becoming commercially successful: the isolation of the supporting tools from the mainstream of commercial software development, and the challenge of defining a large vocabulary of words and phrases by people who are not linguists or logicians. Fortunately, the second challenge can be addressed with freely available resources, such as WordNet, whose terms have been aligned to the major ontologies that are being developed today. The challenge of integrating all the tools used in software design and development with controlled NLs is not a technical problem, but an even more daunting problem of fads, trends, politics, and standards.

Although controlled NLs are easy to read, writing them requires training for the authors and tools for helping them. Using the logic generated from controlled NLs in practical systems also requires tools for mapping logic to current software. Both of these tasks could benefit from applied research: the first in human factors, and the second in compiler technology. An example of the second is a knowledge compiler developed by Peterson et al. (1998), which extracted a subset of axioms from the Cyc system to drive a deductive database. It translated Cyc axioms, stated in a superset of FOL, to constraints for an SQL database and to Horn-clause rules for an inference engine. Although the knowledge engineers had used a very expressive dialect of logic, 84% of the axioms they wrote could be translated directly to Horn-clause rules (4667 of the 5532 axioms extracted from Cyc). The remaining 865 axioms were translated to SQL constraints, which would ensure that all database updates were consistent with the axioms.

In summary, logic can be used with commercial systems by people who have no formal training in logic. The fads and fallacies that block such use are the disdain by logicians for readable notations, the fear of logic by nonlogicians, and the lack of any coherent policy for integrating all development tools. The logic-based languages of the Semantic Web are useful, but they are not integrated with the SQL language of relational databases, the UML diagrams for software design and development, or the legacy systems that will not disappear for many decades to come. A better integration is possible with tools based on logic at the core, diagrams and controlled NLs at the human interfaces, and compiler technology for mapping logic to both new and legacy software.

References

For further discussion and references concerning topics mentioned in this article, see the slides by Sowa (2006a,b).

Limber, John (1973) "The genesis of complex sentences," in T. Moore, ed., *Cognitive Development and the Acquisition of Language*, Academic Press, New York, 169-186. http://pubpages.unh.edu/~jel/JLimber/Genesis_complex_sentences.pdf

Peter of Spain or Petrus Hispanus (circa 1239) *Summulae Logicales*, Marietti, Turin, 1947.

Peterson, Brian J., William A. Andersen, & Joshua Engel (1998) "Knowledge bus: generating application-focused databases from large ontologies," *Proc. 5th KRDB Workshop*, Seattle, WA. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-10/

Sowa, John F. (2003) "Laws, facts, and contexts: Foundations for multimodal reasoning," in *Knowledge Contributors*, edited by V. F. Hendricks, K. F. Jørgensen, and S. A. Pedersen, Kluwer Academic Publishers, Dordrecht, pp. 145-184.

Sowa, John F. (2006a) "Concept mapping," slides presented at the AERA Conference, San Francisco, 10 April 2006. http://www.jfsowa.com/talks/cmapping.pdf

Sowa, John F. (2006b) "Extending semantic interoperability to legacy systems and an unpredictable future," slides presented at the Collaborative Expedition Workshop, Arlington, VA, 15 August 2006. http://www.jfsowa.com/talks/extend.pdf

Wang, Hao (1960) "Toward mechanical mathematics," *IBM Journal of Research and Development* **4**, pp. 2-22. http://www.research.ibm.com/journal/rd/041/ibmrd0401B.pdf