FiledOn: {phylum}<bobrow>papers>mcd-3.tedit
Last Edited: 12 January 1987

Searching Beyond Reason

Comments on McDermott's "Critique of Pure Reason"

by
Daniel G. Bobrow and Mark J. Stefik
Intelligent Systems Laboratory
Xerox PARC

In his paper, McDermott worries that the logical account of deduction doesn't do justice to human reasoning. He sounds like a child who has come to believe that there is no Santa Claus. He has the reasons and rehearses them, but does not want to believe them. So he makes a plea for people to find flaws in the reasons. (Perhaps there IS a way for Santa to fit down radiator pipes.) In contrast, we believe that McDermott does not go far enough in describing the problems with logic. But, to push a metaphor too far, we need not throw out gift giving even if we think that the man in the red suit is an imposter.

We describe how McDermott's critique of the meta-theory defense of logic is somewhat misaimed, and also falls short of the real problems. Logic is expected to be sound, expressive, additive, computationally effective, and fun. We argue that logic is a good thing, but one mustn't expect too much from it. Better results may be obtained by combining logic with principled computational notions and technology, and use logic for what it was designed, a language that shows how a line of reasoning leads soundly to a particular result, rather than as a basis for finding that line.

Logic and Leverage

We begin as does McDermott with the importance of knowledge in reasoning. In McDermott's words:

> "I should first outline the logicist position. It starts from a premiss that almost everyone in AI would accept, that programs must be based on a lot of knowledge."

This leads quickly to some issues about the kinds of knowledge that are needed. McDermott illustrates this in an example about a paper clip and a key chain. In this example, he wants to represent the reasoning that it takes to determine whether a paper clip could be used as a key chain, falling back on various kinds of knowledge about topology, geometry, and perhaps the properties of materials. Would the wire of a paper clip of a certain size bend and fit through the hole of a key? Representations are needed for the necessary reasoning about wires fitting through holes, wires bending, keys slipping past tight spots, keys being fastened securely in a bunch, typical sizes and shapes of keys, and so on. McDermott's bug is that he wants to get all of this using only logic.

There is a kind of additivity assumption that underlies this desire to use logic. In order to start with general knowledge about topology, geometry, and materials, and have them work together without special effort, all of them should be expressed in the same language (logic), and which therefore somehow implies that they can all be thrown into the same pot, and make the right stew. In principle then, one would like to just state the facts ("The facts, ma'am, just the facts" -- Jack Webb) and rely on logic (meaning deduction) to make it all work together.

But McDermott recognizes that this might not quite work. One needs to make the connections between subtheories, and search through the facts of these theories to generate any particular conclusion. McDermott sees the problem but worries about the use of the "meta-theory" which tells how to use the knowledge. He says:

> "There is nothing to say in general about the meta-theory idea; and for any given case there is too much to say ... this study will dwarf the meta-theory framework. You will have to construct a very complex and detailed model to make any progress.

We agree. Of course meta-theories need to be rich. McDermott simply fails to look for a handle on building rich meta-theories. There is no escaping the complexity. Meta-theories themselves will need to be knowledge intensive.

McDermott further suggests that the business of meta-theories is what he calls "legal interventions in object theories." This comes from looking too intensely at deduction as the kernel and framework for problem solving. If deduction is the right stuff, what else could one do besides interrupt it with higher level deductions? The alternative, McDermott fears, leaves us with just programming:

> "If the enterprise becomes one of crafting meta-theories of ... arbitrary power then we might as well admit that we are programming after all."

The problem with programming, he implies, is that it must be completely unprincipled. But, by capturing appropriate abstractions in a programming language, one can (if we are good programmers) find both principled and general ways of handling control. There is much work going on now on architectures that have the nature of a meta-theory, but which attempt to follow programming principles. Brian Smith's 3-Lisp [Smith, 1984] lays out some of the principles and a notation by which interpreters operate at different levels. Stefik [Stefik, 1981] experimented with the notion of strict layers of control, each with its own search space in his Molgen system. Wilensky [Wilensky, 1983] proposed an approach by which the knowledge used at different layers of control could be shared -- when it corresponded to generic ways of handling principled kinds of interactions among different goals. Laird, Rosenbloom, and Newell [Laird, Rosenbloom, & Newell, 1986] have recently re-examined search, seeing the advantage of re-casting the families of search methods into a "universal weak method" in which the special case searches are an emergent phenomenon of the kinds of knowledge that can be applied.

McDermott advocates unprincipled programming with a meta-program that just interrupts and fixes bad deduction. It has two problems. First, as McDermott says, it inhibits precious little; it may not even preserve logic soundness. Secondly, as he doesn't say, it provides precious little leverage. Logic provides exactly one kind of leverage. Making soundness explicit. It is unreasonable to expect that leverage from soundness will be adequate to cope with unlimited potential complexity. "AI programs are notorious for being impenetrably complex." So, of course, logic can be at best only part of the picture. This is where we part ways with McDermott.

Beyond Deduction

Deduction does not address the problem of choosing alternative steps in generating a plan. It provides syntactic criteria for soundness but no process for generating solution candidates. Nor does deduction provide a mechanism for evaluating the relative goodness of alternative candidates. The right next step towards better understanding of reasoning in AI is the elaboration of different forms of search as knowledge-guided process models. We need concepts that will do justice to the richness of human reasoning; concepts from which we can build computational models of problem-solving; models for which we can give a principled account of how they can be guided by knowledge.

In the 1970's it was widely believed that the key to intelligent systems was in the design of general search methods. Much of the motivation for the activity and analysis of different varieties of search -- depth-first versus breadth-first, one-way versus two-way, top-down versus bottom-up -- was the hope that it would lead to the discovery of an exceptionally powerful search method. Such a method failed to materialize.

When no general and powerful search method was found, and expert systems began to succeed at solving problems that were widely recognized as being difficult, there was a shift in perspective. It became better understood that the power of these systems came not from the search methods themselves, but from the knowledge used by the methods. Different kinds of search methods accommodate different kinds of knowledge. This is why the study of search methods retains its relevance. Search methods are procedures, organized to make use of particular kinds of knowledge. Some of the basic search methods can make use of knowledge about such things as closeness to a goal or estimates about path length.

The real opportunities for integrating knowledge into a search process come when large problems are divided into many smaller problems. In this case, it is often appropriate to use one search method for one part of a problem and another search method for a different part. Thus it is usually inappropriate to think of a problem-solving system as employing a single search method: special case search subprograms can be tailored for specific subtasks. For efficiency, search can be organized hierarchically. A knowledge system that performs many subtasks, each with its own search space, is a sort of "conceptual coat rack" offering many places to put special case knowledge where it can be applied. Several of the next generation of knowledge representation shells for building expert systems are based on the composition of basic search methods. [Mittal&Arraya, 1986, McDermott, 1986].

The conceptual coat racks that emerge from the composition of search methods provide us with places to put knowledge. Such systems are just a partial solution. They don't provide additivity in the sense that you can just toss the knowledge in and have the system organize itself. You have to put each piece of knowledge in its place and the frameworks provide a discipline for the placement. In a sense, these frameworks work because the places in the coatracks represent the contexts for use. Learning such contexts for use is the primary purpose of problem exercises in texts, after the principles have been elaborated in a more context free manner.

Clancey's articulation of "Heuristic Classification" [Clancey, 1985] generalizes the search strategies used in EMYCIN and so many of its commercial clones. It is a good example of the kind of model we have in mind for formalization and study. Its process model expects knowledge in a certain form, and processes it according to certain rules. One can study the limits of such a model, e.g., what kinds of knowledge are additive, what kinds of proofs does it generate, what are the average and worst case performances, etc. Models like Heuristic Classification serve a role in knowledge engineering that is analogous to the role that linear programming systems and optimization methods play in operations research. Each establishes methods for solving classes of constraint problems according to certain kinds of solution criteria.

Multiple Representation and Knowledge Compilation

Another problem with using deduction turns on the question of efficiency. In most AI systems, a technique to implement efficient deduction is to choose effective representations. A typical approach is to choose an analogical or vivid representation [Levesque, 1986]. Vivid representations are popular because the representation can take a significant part of the deductive load. For example, one can imagine various 3-dimensional modeling representations in which it would be easy to make some of McDermott's topological inferences about the paper clip and the key chain, based on processes similar to occlusion and intersection. As an aside, it is becoming appropriate now to consider the choice of the computational architecture as part of choosing a representation, as the examples of data-parallelism clearly illustrate [Waltz, 1987].

But this business of having special representations is worrisome to McDermott. He likes efficiency, but he wants to make sure that deduction can be called upon at least occasionally for inferences outside the key ones that the special representations were designed to make efficient. He thinks (and we agree) that this leads to the use of multiple representations, and there he sees troubles. He says:

> "represented knowledge is to be used in just one way. ...[to be used in two ways, each] fact would have to be represented twice. Surely this is not a pleasant requirement to impose on an intelligent program."

The assumption here is that the representations must be created independently and hence could become inconsistent with each other. But this need not be the case. Knowledge can be processed in forms radically different from the form in which it is acquired. Getting knowledge to be multiply represented for processing is not just a question of getting the right language. It may be a question of the appropriate processing on that language. In particular, this suggests the utility of *knowledge compilation*, and thereby the design of systems which take the responsibility of synchronizing multiple representations as needed.

The search for a single representation in which knowledge can be added simply is a bit of a pipe dream. Nonetheless, this flawed goal has arisen in many visions of building intelligent systems. For example, a story similar to that about deduction could be told about the appeal and ultimate disappointment in schemes for encoding knowledge in terms of production rules. Additivity of knowledge requires more than a simple interpreter; it requires an intelligent agent; it takes knowledge to add knowledge. [Stefik, 1986]

Knowledge, as the word is used in knowledge engineering, refers to codified experience, that is, experience that has been processed, recorded, and made ready for re-use. This statement is a useful definition of knowledge because it connects with the practical intuitions of those who build knowledge systems, the theoretical foundations of knowledge representation and situated action, and the issues and problems that are driving the development of the field. The formulation "codified experience" is relevant to those who build knowledge systems. Experience is generally hard won and valued by those who practice something. Introspecting on experience and articulating what has been learned is hard work. To become knowledge, this experience must be articulated and that requires more than just listing lots of facts, such as the price of eggs or the postal address of Ed Feigenbaum. It requires organizing the statement of experience to guide future action.

Those who build expert systems often find that much of what is needed is not to be found in textbooks. Textbooks are not bad or incorrect; the problem is the great deal of practical material -- obvious to an expert -- never finds it way into textbooks. Most textbook knowledge is too idealized. For applications of real interest, only an expert knows the many details of real problems and the unpublished rules of thumb.

Finding out those things that are missing requires building a program that the expert can interact with, and comparing his or her behavior with the program, to find the knowledge that is missing in the "stupid" program. This directly contradicts McDermott's logicist's assumption that

"... we can and should write down the knowledge that programs must have before we write the programs themselves."

This is like programming in a style in which one must first get the specifications right. Our experience in programming in AI strongly suggests that one needs a more empirical attitude about knowledge and experience. There may be unexpected computational surprises arising from the choice and structure of the vocabulary. Going back to the armchair to first "write down the knowledge" without testing it in use as you go is a retreat from the scientific method to Aristotelian science at its worst. The most fruitful approach is to go around in a cycle. It may be that some people get too distracted by the programming and "the knowledge will be shortchanged", but that is a bug in the people, not the process.

Conclusion

What is needed is to put logic in the place it was originally designed for, as a way for people to look at the output of a reasoning process and to see explicitly the assumptions and steps that lead to a conclusion. It is probably also useful as a way for people from different backgrounds to share some of what they know.

Logic is so often placed on a pedestal that some of us have come to think of it as *the* foundation of AI. Actually AI rests on several foundations: computational models from the study of algorithms, deduction and inference models from logic, problem-solving concepts derived from cognitive psychology and operations research, embedded languages and compilation techniques from formal computational languages, and system development concepts from software engineering. AI is a multi-disciplinary field -- and we must combine in a principled way many perspectives on programming [Bobrow&Stefik, 1986] to do the hard job we have set for ourselves.

Acknowledgements

Portions of this paper were drawn from "Introduction to Knowledge Systems", a textbook in preparation by Mark Stefik.

References

[Bobrow&Stefik86] Bobrow, D.G., and Stefik, M. J. Perspectives on Artificial Intelligence Programming. *Science* **231:4741**, pp. 951-956, 28 February 1986. (Reprinted in Rich, C. & Waters R.C. (Eds.) *Readings in Artificial Intelligence and Software Engineering*, pp. 581-587, Los Altos: Morgan Kaufman Publishers, 1986.)

[Clancey85] Clancey, Wiliam J., Heuristic Classification *Artificial Intelligence*, 27:3, pp. 289-350, December 1985

[Laird, Rosenbloom & Newell, 1986] Laird, J., Rosenbloom, P., & Newell, A. Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies. Boston, Massachusetts: Kluwer Academic Publishers, 1986.

[Levesque86] Levesque, Hector, Making Believers out of Computers, *Artificial Intelligence*, 30:1, pp. 81-108, October 1986

[McDermott86] McDermott, J., Making Expert Systems Explicit, *Proceedings of the IFIP Conference*, Dublin, Ireland, 1986

[Mittal&Arraya86] Mittal, S. & Arraya, A. A Knowledge-based Framework for Design, *Proceedings of the AAAI-86 Conference*, p856-865, 1986

[Smith84] Smith, Brian C., Reflection and Semantics in Lisp, *ACM Conference Record of the 11th Principles of Programming Language Conference*, Salt Lake City, Utah 1984, p23-35

[Stefik81] Stefik, M., Planning and Meta-planning (MOLGEN Part 2). *Artificial Intelligence*, 16:2, pp. 141-169, May 1981. (Also reprinted in N. Nilsson & B. Webber, *Readings in Artificial Intelligence*, Tioga Publishing Company, 1982)

[Stefik86] Stefik, M., The next knowledge medium. *AI Magazine* 7:1, Spring 1986.

[Waltz87] Waltz, David L., Applications of the Connection Machine, *IEEE Computer*, 20:1, pp 85-99, January 1987

[Wilensky83] Wilensky, R. Planning and Understanding: A Computational Approach to Human Reasoning. Reading, Mass.: Addison-Wesley, 1983.