Machine Learning, 4, 337–338 (1989)

The World Would Be a Better Place if Non-Programmers Could Program

JOHN McDERMOTT
Digital Equipment Corporation, 290 Donald Lynch Boulevard, DLB5-3/E2, Marlborough, MA 01752

Each task that anyone, man or machine, might want to perform imposes some set of computational requirements on the performer. This is obvious—how could it be otherwise? But if you ask anyone what the computational requirements are that a class of tasks imposes, you surely won't get a very good answer. As application programmers, we can expose the computational requirements of the task we just wrote a program to solve. But we typically do so by pointing to the program; this doesn't provide much insight, either to ourselves or to others, into the requirements that other, similar tasks impose. From the perspective of one who simply wants an understanding of the computational requirements of a class of tasks, the application program over-commits.

Over the years, AI researchers have had insights that can potentially improve this state of affairs. In particular, the idea that an inference engine (a control structure) can be defined that operates on domain knowledge to solve problems is a good starting point. An inference engine is precisely the set of mechanisms that satisfies the computational requirements of some class of tasks. Thus showing someone an inference engine, rather than a complete program, at least has the value that it is appropriately general. But there is still a problem. We don't have concepts that allow the computational requirements of tasks to be easily discussed. Another way of saying this is that no inference engine builder has yet been successful at defining, in terms understandable to those familiar with tasks, the characteristics a task has to have in order for it to be an appropriate task for his or her inference engine.

If we had such a set of concepts, we could develop a wide variety of useful and effective inference engines. Each engine would provide the computational mechanisms required to address a particular class of tasks—i.e., all tasks with a specific set of characteristics. Since each set of mechanisms presupposes the availability of certain types of information, the mechanisms effectively define the knowledge that has to be elicited from those familiar with the task. Moreover, since each set of mechanisms requires particular access paths to that information, the mechanisms effectively define how the knowledge can be appropriately represented. Thus such engines should substantially simplify the program development and maintenance process.

It should be easy to begin to invent the appropriate set of concepts. A task has characteristics that need attention whenever it is performed. For example, for a subset of diagnostic tasks, a critical task characteristic is the existence of competing possible explanations that must be discriminated among. Or for a subset of configuration tasks, a critical task characteristic is the existence of an ordered set of fixes that can be used to modify configurations that violate constraints. I'll call the concept that acknowledges the diagnostic task characteristic the "competing explanations" concept. And I'll call the concept that acknowledges the configuration task characteristic the "obvious fix" concept. A set of mechanisms suggest themselves for each of these two characteristics.

For tasks involving competing explanations, two mechanisms suggest themselves: (1) a mechanism for identifying candidate explanations, and (2) a mechanism for evaluating candidate explanations relative to one another. For tasks involving obvious fixes, four mechanisms suggest themselves: (1) a mechanism that keeps track of component choices and why they were made (i.e., a dependency network), (2) a mechanism for identifying constraint violations, (3) a mechanism for retracting components that violate constraints, and (4) a mechanism for selecting components to substitute for retracted ones. The trick, of course, is getting the level of abstraction right so that no mechanism has to be further specialized in order to deal appropriately with a task instance, while at the same time each set of mechanisms comprising an engine covers as many task instances as possible.

For example, to say that the competing explanation characteristic requires a mechanism to evaluate candidate explanations is, of course, not an adequate specification since there are a number of such mechanisms more or less appropriate for different diagnostic tasks. But the important thing to note is that the number of mechanisms that produce interestingly different results is small. It matters whether the mechanism uses rule-in or rule-out as its primary strategy. It matters whether the mechanism treats evidence as having a likelihood or as having a threshold of relevance. It may matter whether evidence is arithmetically combined or not. And it matters whether the order in which questions are asked is determined by the expected cost of obtaining the information or by the expected value of the information. But for this mechanism, that's about all that matters. My bet is that getting to an appropriate level of abstraction for all mechanisms for all tasks requires specializations of approximately the same degree.