A critique of pure reason[1]

DREW MCDERMOTT
Yale University, New Haven, CT 06520, U.S.A.

Comput. Intell. 3. 151–160 (1987)

In 1978, Patrick Hayes promulgated the *Naive Physics Manifesto*. (It finally appeared as an "official" publication in Hobbs and Moore 1985.) In this paper, he proposed that an all-out effort be mounted to formalize commonsense knowledge, using first-order logic as a notation. This effort had its roots in earlier research, especially the work of John McCarthy, but the scope of Hayes's proposal was new and ambitious. He suggested that the use of Tarskian semantics could allow us to study a large volume of knowledge-representation problems free from the confines of computer programs. The suggestion inspired a small community of people to actually try to write down all (or most) of commonsense knowledge in predictate calculus. He launched the effort with his own paper on "Liquids" (also in Hobbs and Moore 1985), a fascinating attempt to fix ontology and notation for a realistic domain. Since then several papers in this vein have appeared (Allen 1984; Hobbs 1986; Shoham 1985). I myself have been an enthusiastic advocate of the movement, having written general boosting papers (1978) as well as attempts to actually get on with the work. (1982, 1985). I even coauthored a textbook oriented around Hayes's idea (Charniak and McDermott 1985).

It is therefore with special pain that I produce this report, which draws mostly negative conclusions about progress on Hayes's project so far, and the progress we can expect. In a nutshell, I will argue that the skimpy progress observed so far is no accident, that in fact it is going to be very difficult to do much better in the future. The reason is that the unspoken premise in Hayes's arguments, that a lot of reasoning can be analyzed as deductive or approximately deductive, is erroneous.

I don't want what I say in this paper to be taken as a criticism of Pat Hayes, for the simple reason that he is not solely to blame for the position I am criticizing. I will therefore refer to it as the "logicist" position in what follows. It is really the joint work of several people, including John McCarthy, Robert Moore, James Allen, Jerry Hobbs, Patrick Hayes, and me, of whom Hayes is simply the most eloquent.

The logicist argument

I should first outline the logicist position. It starts from a premise that almost everyone in AI would accept, that programs must be based on a lot of knowledge. Even a program that learns must start out knowing many more facts than it will ever learn. The next step is to assume that this knowledge must be represented somehow in the program. Almost everyone in AI would accept this, too, but with subtle reservations, which I will come back to later in the paper.

The next step is to argue that we can and should write down the knowledge that programs must have before we write the programs themselves. We know what this knowledge is; it's what everybody knows, about physics, about time and space, about human relationships and behavior. If we attempt to write the programs first, experience shows that the knowledge will be shortchanged. The tendency will be to oversimplify what people actually know in order to get a program that works. On the other hand, if we free ourselves from the exigencies of hacking, then we can focus on the actual knowledge in all its complexity. Once we have a rich theory of the commonsense world, we can try to embody it in programs. This theory will become an indispensable aid to writing those programs.

The next step is to argue that mathematical logic is a good notation for writing the knowledge down. It is in some sense the *only* notation. The notation we use must be understandable to those using it and reading it; so it must have a semantics; so it must have a Tarskian semantics, because there is no other candidate. The syntax of the notation is unimportant, so let's use a traditional logical notation, because we already know how to extend it as far as we're likely to go. The supposed advantages of newer notations (e.g., semantic networks) are based on confusion and fantasy — confusion about implementation versus content, and fantasy about what we sometimes wish our notations would mean.

It is possible to misunderstand the role that logic is supposed to play in the programs that we will ultimately write. Because we started by saying that knowledge ought to be represented, one might conclude that the axiomatic theories we construct will eventually appear explicitly in the programs, accompanied by an interpreter of some kind that reads the axioms in order to decide what to do. This model may in fact be realized, or it may not be. It is possible, for instance, that a transitivity axiom will not appear explicitly in a program, but may be embodied in some kind of graph traverser. The logicists' argument is that we ought to forget about the grubby details of programming for a while (for a generation, let's say) and instead write down what people know (or coherently believe) about everyday life. If we can actually come up with a formal theory of informal knowledge, then we will have something that future programmers will have to reckon with — literally. From this point of view, it is all the clearer why logicists favor classical logic over newer notations like associative networks. The purported differences between logic and the other notations is that the latter organize knowledge in various ways for use by programs, and that is precisely what the logicists are not interested in. All they want are the facts, ma'am, facts like "liquids leave tilted containers." Unfortunately, this is not the only motive for favoring logic. There is an unspoken premise in the argument that a significant amount of thought is deductive. Without this premise, the idea that you can write down what people know without regard for how they will use this knowledge is without foundation.

How are we supposed to know when we have made progress in formalizing knowledge? Suppose we write down a bunch of axioms. How will we know when we've written down most of

[1]This paper is an amplification of a talk given at the AI Society of New England meeting, November 1985. The paper contains a few masculine pronouns that some might object to; bear in mind that the use of such a pronoun to refer to an anonymous person does not preclude his being female.

what people know about the subject? Well, when we can't think of anything else to say. But people aren't that great at writing down everything they know about a subject; they tend to leave things out. How will we know when we're really getting there? I think the logicists take it for granted that we'll be done when all the straightforward inferences follow from the axioms that have been written down. If something is obvious to people, Hayes says somewhere, then it must have a short proof. For this dictum to make sense, a substantial portion of the inferences people make must be deductive. Water flows out of a tilted cup; this cup is tilted; ergo, water will flow out of it. If most inferences fit the deductive pattern, then the notion of logical proof provides an idealized model of inference. We don't have to have any particular process model in mind, because *every* process model we devise will be an approximation to this ideal inference engine, and will have to conform to the inferences that it licenses. When we design an inference mechanism, it can in fact record the reasons for its inferences in terms of the deductive support that they receive. This is the idea behind data dependencies (Doyle 1979): a program may use any method to arrive at a conclusion, but it ought to be able to list all the actual premises that justify it, so that if any of those are erased, the conclusion will be erased, too. The idealized inference engine justifies the practical inference engine.

Defending deduction

But many inferences are not deductive. If I come upon an empty cup of soda pop that was full a while ago, I may infer that my wife drank it, and that's not a deduction (except in Sherlock Holmes's sense), but an inference to the best explanation. (The only way to mistake this for a deduction is to mistake logic programming for logic; more on this below.) If almost all inferences fall into this or some other nondeductive category, the logicist program will be in serious trouble. It must be the case that a significant portion of the inferences we want are deductions, or it will simply be irrelevant how many theorems follow deductively from a given axiom set. Whatever follows deductively would in that case be *ipso facto* trivial.

Unfortunately, the more you attempt to push the logicist project, the less deduction you find. What you find instead is that many inferences which seem so straightforward that they *must* be deductions turn out to have nondeductive components. We can begin with two examples from Hayes's (1985*b*) Liquids paper:

> Suppose an open container with no leaks is empty, but at time *t* a falling history begins whose *bottom* is the free top of the container: for example, you turn on the bath tap with the plug in. By axiom (46), this leaving has an arriving on its other side, which is an inward-directed face of the inside of the bath. By axiom (59), there must be a filling inside the bath, so the *amount* of water increases: axiom (61). So long as the tap keeps running, it will go on increasing. Let us suppose that eventually the bath is full, i.e., it contains its capacity. [So the bath will overflow.] (Notice that if the container were closed—a tank being filled along a pipe, say—then the same line of reasoning would insist on there being a leaving which could not possibly occur . . . One can conclude from this contradiction . . . that the arriving must cease to exist at that time, and hence that the flowing . . . along the supply pipe . . . must cease also . . .)

This seems like a beautiful pair of arguments, a perfect illustration of Hayes's desideratum that obvious inferences have short proofs. Unfortunately, though they may be short, they are not proofs. Suppose we grant Hayes's analysis of the second case; we made the assumption that the filling lasted a certain amount of time, got a contradiction, and concluded it would not, after all, last that long. But then the first case, if we are to follow anything like uniform rules, must be a case of making an assumption and *not* getting a contradiction. All right, but you are not allowed in deduction to make an argument of the form "Assume *P*; no contradiction?; okay, conclude *P*." Something else is going on.

Another example is the treatment of the planning problem by logicists, such as Rosenschein (1981). They cast the problem thus: given some axioms about the effects of actions in the world, an initial state of affairs, and a desired state of affairs, find a sequence of actions that can be proven to transform the world from the initial state to the desired state. This is an interesting problem, but it has nothing to do with planning as practiced by corporate managers, ordinary people, or robots. Think of the last time you made a plan, and ask yourself if you could have *proven* the plan would work. Chances are you could easily cite ten plausible circumstances under which the plan would *not* work, but you went ahead and adopted it anyway. In fact, all of the hard parts of planning — especially replanning during execution — are incompatible with the view that the object is to prove a plan correct.

This informal survey is borne out by the meager results that the logicists, including me, have had. In case after case, what can actually be written down as axioms is fairly puny. (I forbear from citing other people's work in this context; one example of mine is McDermott 1985). On the other side, one finds nonlogicist researchers like Forbus who concentrate on writing algorithms to draw inferences, but let themselves be intimidated by the logicists into thinking they really should be able to express as axioms the content of the knowledge in those algorithms. The results (e.g., the axioms in Forbus 1984) are silly, and fall way short of expressing what they are supposed to. I used to think the failure was Forbus's, but I would now exonerate him, and blame the task, seemingly so feasible but actually impossible.[2].

The obstacles I am describing are not news to logicists. From the beginning, it has been clear that the logicist project had to be qualified, or specially interpreted. In what follows, I will describe all the known defenses of logicism, and argue that they all fail. There defenses are not mutually exclusive; each compensates for a different set of ailments, and most logicists have probably believed in most of them most of the time. Here is the list:

1. The "idealization" defense: View deductive formulations of problems as idealizations. The deductive planning problem, for instance, can be seen as an idealized version of the "real" planning problem, carried on, perhaps, in a world theory that is an idealized version of the real-world theory.

2. The "vocabulary" defense: Emphasize that we can pick whatever pedicates we like. It may be true that we cannot deduce that a particular plan will work, but if we change the problem to one of deducing should-do(*agent*, *plan*), then progress will be easier.

3. The "queen of the sciences" defense: Find fertile bonds between deduction and nondeductive inference. For instance, inference to the best explanation can be seen as finding

[2]When I say Forbus was "intimidated," I mean it literally. I refereed his paper, and asked him to try to be more logicist. Mea culpa.

premises from which an observed conclusion follows deductively. In this way deduction comes to be the centrepiece of a grand theory of reasoning, surrounded by interesting variants of deduction.

4. The "metatheory" defense: Posit the existence of deductive "metatheories," theories about how to find and edit the conclusions of the original, defective "object level" theories.

5. The "deducto-technology" defense: Argue from the existence of logic programming that many realistic inference problems can be seen as essentially deductive.

6. The "nonmonotonic" defense: Argue that by extending classical logic to allow defeasible conclusions we can capture a significantly larger set of inferences.

I will refute each of these in order, starting with the idealization defense.

Idealizations are not always bad; they are often essential. For instance, it may be a useful idealization to prove that a certain plan will win a game of chess, even though the proof neglects the possibility that someone might suddenly offer the planner a million dollars to throw the game. One is entirely justified in simply leaving that kind of possibility out of the axioms. However, what I am arguing against is a mentality that would assume in all chess situations that the goal is to find a provably winning strategy, or that would overlook more normal situations in favor of situations where such a proof was possible. The fact is that realistic chess programs (and human players) do nothing remotely resembling proving that a plan will work. Of course, for any given algorithm, say, game-tree search, there is a way of viewing what it does as deducing *something* (e.g., the minimax value of a tree), but this claim is of no interest to us.

I am afraid that in many cases where a deductive problem is claimed as an approximation to a realistic problem, it is actually an *analogue* to a realistic problem, the best deductive mimic of the real thing. In many cases, this fact may not be insuperable. The attempt to write down facts in the analogue domain may yield insights into the actual domain. The resulting ontology and axioms may be useful for eventually writing programs. What we cannot expect from an idealization is the *coverage* the logicists are expecting. Many concepts from the real domain will just not be found in the idealization. Contrariwise, there is the danger that too many concepts from the ideal domain will not be found in the real one, and the idealization will be so askew as to be useless. Still, as a strategy the use of idealizations does seem worthwhile, and I will come back to this idea at the end of the paper.

Next, the "vocabulary" defense. The point made here is certainly one I would embrace. If one is designing a program to think about mathematics, being committed to a deductive approach does not entail confining the vocabulary of the program to Zermelo–Frankel set theory. Instead, one would want whatever predicates a human mathematician would use, such as interesting_concept(*C*), appears_provable(*theorem*), and so forth. For instance, one might take all the predicates used implicitly by Lenat in AM (Lenat 1982) and try to write a program that deduced that a concept was interesting or a theorem was probably provable.

The problem with this defense is that so far it hasn't helped. By broadening the range of problems that can be cast as deduction, we have in many cases simply added to the list of problems we can't solve using deduction. There are good reasons by AM was not a deductive program.

Another problem with the "vocabulary" defense is that it allows us to replace a hard problem with a trivial one. For instance, in medical diagnosis, if we run into trouble deducing diagnosis(*patient, disease*), switching to possible_diagnosis(*patient, disease*) is not going to help. The new problem in this case is too easy; all of the action is in differential diagnosis and weighing evidence, which will now be neglected, or passed off to some nondeductive module.

The "queen of the sciences" defense may be elaborated thus: Consider "abduction," C. S. Peirce's term for explanatory hypothesis generation. This process is nondeductive, but we can think of it as a sort of "inverse deduction." For instance, to explain *q*, look for an implication of the form (if *p q*) that you already know, and propose *p* as an explanation. Put more generally, to explain *q*, find premises that combined with what you already know will entail *q*. If this model is correct, then even though abduction is not a kind of deduction, still it is justified by deduction. (This view is endorsed, with reservations, by Charniak and McDermott (1985).)

This account of explanation is known among philosophers as the *Deductive-Nomological Theory*. It is most commonly associated with the name of C. G. Hempel (Hempel and Oppenheim 1948; Hempel 1965). Unfortunately, it is believed by almost no one else. It has several bugs as a model of scientific explanation, which is what it was devised for, and seems hopeless as a model of explaining the behavior of individual humans or physical systems. The problem is that a deductive chain between *explanans* and *explanandum* is neither necessary nor sufficient.

One reason it is not necessary is that we are content if the explanation merely makes the observed facts probable. Hempel allowed for this case, and so do all the diagnostic expert systems like Mycin (Shortliffe 1976) and Prospector (Duda *et al.* 1980).

But there are more devious examples. Suppose you read in the paper that Selma McGillicuddy, of Secaucus, just won the New Jersey lottery for the second time in the last two months, for more than a hundred thousand dollars each time. There is no reason to infer corruption, so you arrive at the explanation, *It's a fair lottery; occasionally someone will win twice in succession.* This is a satisfying explanation, but you cannot infer from it, "Selma McGillicuddy wins the lottery twice in two months." (Wesley Salmon first pointed out this class of explanation; see Salmon (1967, 1975).)

The reason why a deduction of the data is not sufficient is that the requirement is too easy to meet. There will in general be millions of deductions leading to the observed conclusion, almost all of which are absurd as explanations. For example, one day I noticed that my clock radio was two minutes fast. Since I am compulsive about accurate clocks, I was bothered, and sought an explanation. It occurred to me that there had been a power failure lasting two hours recently. One would therefore expect that the clock would be two hours slow, but I remembered that it had a battery backup clock. Hence, the proper explanation was that the battery-powered clock was inaccurate, and gained about a minute an hour.

Let us assume that this explanation could be turned into a deductive argument, with the conclusion, "The clock is two minutes fast." So what? There are plenty of other deductions with the same conclusion. ("A visitor to our house maliciously set the clock ahead." "A burst of cosmic rays hit the clock just right.") One can argue that these are obviously inferior explanations, and all we could hope for from the "Queen of the Sciences" picture is a characterization of an *adequate* explanation, but this is a sterile position. The condition of adequacy is just too trivial.

If we are not careful, it can become even more trivial. A premise like "Every clock in the room is two minutes fast" will explain "This clock is two minutes fast," if this clock is the only one in the room. Hempel sought to avoid this problem by requiring the premises and conclusion to be "lawlike." It is not really clear what this property amounts to, but it is intended to rule out "The number of planets is the least odd square of a prime number" as an explanation for "There are nine planets." In fact, it eliminates from consideration any explanation of a particular fact, and makes the theory into a theory of explaining laws.

In a way, the "Queen" defense is a version of the "idealization" defense, with similar weaknesses. There must be *some* link between the hypothesis and the evidence to be explained, but it is merely dogmatic to surmise that the link is deductive. In general, about all we can say about it is that a good hypothesis is one that satisfies a typical human inquirer. I will come back to this topic later. For now, we can conclude that deduction cannot be the centerpiece of a theory of abductive inference.

The "metatheory" defense argues that the problems with a deductive inference engine can be fixed via the intervention of a deductive "meta-engine" that steps in and edits its output; or alters its premises; or turns it off to allow a competing theory to take over. For instance, in the two tank examples from Hayes, we can imagine this meta-engine doing *belief revision*, introducing premises about the persistence of flows, and retracting them when awkward conclusions arise.

The problem with this defense is its vacuity. The subject matter of the deductive metatheory must presumably be "legal interventions in object theories." But there are no constraints on such a theory, from human intuition or anywhere else. There is certainly no constraint that the interventions preserve deductive soundness. If there were, this defense would not accomplish the required strengthening of deduction. So it is difficult to see how to rule out a theory like "Believe all statements with an odd number of symbols on weekends; believe all statements with an even number on weekdays." If the enterprise becomes one of crafting metatheories of such arbitrary power, then we might as well admit we are programming after all.

There is nothing to say *in general* about the metatheory idea; and for any given case there is too much to say. Let's take the idea of *belief revision*, which I bandied about a couple of paragraphs back. If you start studying this seriously, you eventually wind up studying nonmonotonic logic (about which much more below). This study will dwarf the metatheory framework. You will have to construct a very complex and detailed model to make any progress, and long before you are done it will be clear that it is completely irrelevant whether it is targeted for implementation as some kind of deductive metatheory or instead as a Lisp program. The metatheory framework contributes nothing, unless you just prefer Prolog to Lisp.

This brings me to the fifth defense, "deducto-technology." One reason it is easy to overestimate the power of deduction is because of the existence of a powerful set of tools, such as backward chaining and unification, which are derived from automatic-theorem-proving research, but have found a wider popularity in systems like Prolog (Clocksin and Mellish 1981) and MRS (Genesereth 1983). These tools turn out to provide an elegant model of computation, just as powerful as, and in some cases prettier than, traditional models. Because you can use these tools to do any computation, and because of their genesis in theorem provers, it is natural to draw the conclusion that any computation is in some sense deduction. It is difficult to refute the argument leading to this conclusion, because there is no argument as such, just vague associations among concepts. (The fallacy is certainly not hindered by the use of phrases like "logical inferences per second" by the logic-programming community to refer to something as trivial as list-processing operations.) Serious researchers are not consciously taken in by the fallacy, but even they can get carried away by the cleverness of deducto-technology.

To take one example of the sort of woolly thinking we are up against here, consider the way in which values are computed in Prolog-type systems. A goal containing variables is interpreted as a request to find values for the variables. The goal append([a, b], [c, d], X) means, "Find an X that is the result of appending [a, b] and [c, d]." If the axioms are written right values will be found; in this case, X will get bound to [a, b, c, d]. Contrast this goal with append([a, b], [c, d], [a, b, c, d]), where the goal is to verify that [a, b, c, d] is the result. It is a property of reasonably well-behaved Prolog programs that whenever they can find a value they can verify it. (The opposite property is much harder to achieve. Shoham and McDermott 1984).

The problem is that while the idea of verifying conclusions carries over to deduction in general (since it's just the idea of proving something), the idea of calculating values does not. From the point of view of logic, append([a, b], [c, d], X) is just a Skolemized version of (not(exists(X), append([a, b], [c, d], X))). (See any textbook for an explanation of Skolemization, and of the "not.") It is essentially a useful accident that backward chaining verifies this conclusion by finding a value for X. If we try to generalize beyond backward chaining, the idea falls apart. Luckham and Nilsson (1971) give a variant that works for any resolution proof, but not every resolution proof generates a single value per variable. More important, once logic is extended beyond finitely axiomatizable first-order theories (and it often is in the representation-of-knowledge business), the whole idea of resolution and Skolemization becomes irrelevant.

Even when the idea works, logic does not provide a general theory of answer construction. Consider Robert Moore's "Bomb in the Toilet" problem: you receive two indistinguishable ticking objects in the mail, plus an anonymous phone call warning that exactly one of them is a bomb. From old movies, you know that putting a bomb in the toilet is a sure-fire way to disarm it. What should you do? The answer is, Put both objects in the toilet. (Perhaps a bathtub would be better.) But if we pose the problem logically as

```
effect(Plan, and(disarmed(object1),
                 disarmed(object2)))
```

we might get back (using Luckham and Nilsson's procedure)

```
Plan = put(object1, toilet)
```

or

```
Plan = put(object2, toilet)
```

That is, the theorem prover will have cheerfully verified that there is a workable plan, without actually constructing one. Of course, we can't really ask any more. Deduction just doesn't provide a theory of computing arbitrary things. All it aspires to is a theory of verifying arbitrary things.

The nonmonotonic defense

Finally, we come to the most potent defense, the appeal to "nonmonotonic logic," the name given to a system of logic in which conclusions can be *defeasible*, that is, subject to withdrawal given more premises. This sort of logic looks tailor-made for examples like the two involving tanks described above. We want to infer that the water is still flowing into the tank so long as we have no reason to believe otherwise; when a contradiction materializes, the conclusion will be withdrawn.

Nonmonotonicity is almost by definition incompatible with deduction. Hence, as Israel (1980) has pointed out, "nonmonotonic logic" is somewhat oxymoronic. It is as if to compensate for some deficiency of prime numbers we were to propose studying "composite primes." In practice, what is meant by "a nonmonotonic logic" is an inference system that provides a simple, general extension to ordinary logic that captures obvious defeasible inferences. We don't expect such a system to do inference to the best explanation, but we do expect it to infer that your car is still where you parked it last.

Since there might be many alternative "simple, general" extensions to ordinary logic, we cannot draw any final conclusions about the prospects for nonmonotonic logic. We can, however, survey what has been accomplished and evaluate its promise for the future. There are two main methods that have been employed, the *default* approach and the *circumscriptive*. The default approach attempts to formalize the "negation as failure" idea of Planner (Hewitt 1969) and Prolog (Clocksin and Mellish 1981). We extend ordinary logic by allowing inference rules of the form "From premise *p* and the inability to infer *q*, infer *r*." The idea is that *r* is the default conclusion from *p* in the absence of special overriding information *q*. An example would be

$$\frac{(\text{bird } a)\ \text{Consistent}(\text{not}(\text{abnormal } a))}{(\text{can\_fly } a)}$$

where "Consistent *formula*" means that *formula* is consistent with all the inferences in the system. For any given bird, we can then normally infer that it can fly, but if there are axioms for inferring abnormality, then we can use them as "gates" to turn this rule off. Systems of roughly this form have been studied by Reiter (1980), McDermott and Doyle (1980), Clark (1978), and others.

The circumscriptive approach, developed by McCarthy (1980) and his colleagues (Lifschitz 1985; Lifschitz, unpublished manuscript[3]), avoids adding new inference rules, and instead augments a first-order theory with an axiom that expresses the goal to "minimize" some predicate. For instance, given an axiom

```
(forall(x)(if(and(bird x)(not(abnormal x)))
        (can_fly x)))
```

one would want to minimize the abnormal predicate, so that as before we can normally infer for any given bird that it can fly. To achieve this, we add to the original theory a second-order axiom. Let A(abnormal;bird) be the conjunction of all the axioms we already have. (It had better be finite.) In the parenthesis following the A we write the names of predicates we intend to substitute for. The semicolon separates the to-be-minimized predicate (abnormal) from the "variable" predicates (bird); there may in general be one or more to be minimized, zero or more variable. So A (foo;baz) would be the same set of axioms with abnormal replaced by foo and bird replaced by baz. Given this notation, the new axiom is

```
(forall(p b)
   (if(and)(A(p;b))
          (forall(x)(if(p x)(abnormal x)))
      (forall(x)(if(abnormal x) (p x)) )))
```

That is, if p is any predicate that satisfies A (after A has been weakened by changing bird) and is as strong as abnormal, then abnormal is as strong as p. Another step is now required, and that is to plug values in for p and b. Suppose that the only bird we know that can't fly is Clyde. Then A will include the axiom about normal bird flight, plus (bird Clyde) and (not (can_fly Clyde)). If we plug in p = (lambda(y)(= y Clyde)) and b = (lambda(y)(= y Clyde)), then, because A(p;b) becomes

```
(and(forall(x)(if(and(= x Clyde)(not(= x Clyde)))
             (can_fly x)))
   (= Clyde Clyde)
   (not(can_fly Clyde)))
```

the instance we get of the circumscription axiom is

```
(if(and(forall(x)(if(and(= x Clyde)(not(= x Clyde)))
             (can_fly x)))
      (= Clyde Clyde)
      (not(can_fly Clyde))
      (forall(x)(if(= x Clyde)(abnormal x))))
   (forall(x)(if(abnormal x)(= x Clyde))))
```

But the antecedent of this implication follows from A (abnormal;bird), so we can conclude the consequent, that Clyde is the only abnormal object. Hence any other bird (if we can show him unequal to Clyde) will be judged able to fly.

Note how circumscription achieves nonmonotonicity. When a new axiom is added to A, the circumscriptive axiom changes, and usually some theorem goes away.

There are two problems with all known varieties of nonmonotonic logic. The first is that it is often not clear without considerable effort what the consequences of a set of rules are. The second is that they often fail to achieve the proper "amplification"; that is, the rules will have overly weak consequences. I will label these two problems with the phrases "You can't find out," and "You don't want to know."

In default logics, the "You can't find out" problem arises because it is in general undecidable whether a formula is consistent with a theory. In fact, it is even hard to define exactly what is meant by the phrase "consistent with a theory," when the theory in question is the one containing the default rules. You can't tell what isn't inferrable until you've inferred everything, and so we are led to the idea of a "stable extension" or "fixed point" of a default theory. Such a fixed point is a set of formulas, intuitively a "stable set of beliefs," which is characterized by a set of "nonconcluded formulas," such that (a) everything in the fixed point follows from the original theory plus the nonconcluded formulas via default inference rules; and (b) no nonconcluded formula follows. If Clyde is known to be a bird, then (not(abnormal Clyde)) will be one of the nonconcluded formulas, and hence (can_fly Clyde) will be an element of the fixed point; unless (abnormal Clyde) is also

[3]Lifschitz, V. 1986. Pointwise circumscription. Unpublished draft, January 16, 1986.

deducible, in which case (can_fly Clyde) will not be in the fixed point. Unfortunately, the fixed points and the sets of non-concluded formulas are infinite, and in general hard to describe or find.

Circumscription is also hard to use. What all known versions of circumscription have in common is this procedure for arriving at conclusions:

—Add second-order axiom to original theory
—Guess predicate constant to plug in to the axiom
—Simplify

This is the kind of procedure we followed in the example. The problem with it is that the information added is usually about the same size as the conclusions you ultimately want to draw. In fact, it usually looks about the same as those conclusions, with a few extra lambda's.

In principle, circumscription could be used mechanically; you could turn a crank and all the conclusions would come out. In practice, there is no way to enumerate useful instances of the second-order axiom, so circumscription has been used only on small examples for which the desired conclusions are already known. (In special cases, you can show that circumscription and default logics both reduce to computable algorithms, but these special cases are of no interest to us here.) Paradoxically, the hopelessly undecidable default logics suggest a practical algorithm that actually gets somewhere: To verify that p is consistent, try to prove its negation and fail. When this procedure halts, it is often a good heuristic approximation (Clark 1978).

The intractability of nonmonotonic logic has led to a curious phenomenon. Logicists go ahead and use nonmonotonic constructs, and state in the accompanying text what conclusions they hope will follow, without really knowing if they will. At this point, it is no longer clear in what sense the reasoning they are describing is justified by a formal system. This wishful thinking wouldn't matter much if the wishes came true. Unfortunately, this brings us to the "You don't want to know" problem: When a nonmonotonic system is studied carefully, it often happens that the conclusions the formal system actually allows are different from, typically weaker than, what was expected. In default formulations, the problem arises because the fixed-points described above are often nonunique. Some of them are reasonable, but many correspond to sets of beliefs that would be rejected by the person writing the original rules. (There is no way to eliminate such fixed point by adding more default rules; that can only make matters worse.) If a theory has several alternative fixed points, what actually can be said to be a theorem of the theory? Either theories like this don't have theorems, in which case they can't serve as the idealized inference engine we are seeking; or we are stuck with a weak notion of theorem, in which a theorem is something that is inferred in all fixed points. Typically this alternative gives us disjunctive theorems, where some of the disjuncts are counter-intuitive intruders from unwanted fixed pints. We want the conclusion p, but we wind up with p *or* q, where q is off the wall.

Such overweak disjunctions pop up in the circumscriptive versions, too. The phenomenon is somewhat different for circumscription, because the notion of fixed point doesn't play the same proof-theoretic role. But we do have a homologous idea, the *minimal model*, defined as follows. One model is "smaller" than another with respect to some predicate P if it agrees on all other (nonvariable) predicates and its P is a subset of the other model's. A minimal model is one with no model smaller. It can be shown that a formula is true in all minimal models of $A(P; V)$ if it follows from the $A(P; V)$ plus the circumscriptive second-order axiom given above.

The overweak disjunction problem now appears in the following form. Typically there will be minimal models that differ in important ways, such that some of the models are "obviously wrong" to a human observer. On the syntactic side, circumscription will yield a disjunction, such that each disjunct characterizes a class of minimal models. Hence the situation is not really that different from the default-logic case, except that the disjunctions come about as a consequence of the basic machinery, rather than being tossed in as a kludgy way of defining the notion of theorem.

In a recent paper, Hanks and McDermott (1985, 1986) explored one instance of this phenomenon in detail. We studied a simplified version of the temporal logic of McDermott (1982), which was somewhat more complex than the previous nonmonotonic systems that had been studied. We were hoping to show that the conclusions we wanted from the formal system really did follow. We expected that the multiple-fixed-point problem would defeat the default logics, but we expected circumscription to work. We were surprised to discover that circumscription had the same problem as the default formulations, although in retrospect the similarities among the various systems seem so overwhelming that the surprise is lessened.

The problem for all the logics is that concepts like "minimization" and "stable sets of beliefs" are just inappropriate for the temporal domain. The nonmonotonic rule we wanted was (to put it informally) "states of the world tend to remain undisturbed." All the logics drew conclusions that minimized disturbances, but that's not what we really wanted. Instead, we wanted to avoid disturbances with unknown causes.

What we were trying to state was that a "history" continues unless it is explicitly "clipped" by subsequent events. Consider the following event sequence:

1. Fred is born	Fred starts to be ALIVE
2. A gun is loaded	Gun starts to be LOADED
3. Fred is shot with it	Fred becomes DEAD

We ought to be able to conclude that Fred is now dead (sorry for the violence). But another scenario would minimize disturbance equally well. In this one, the gun ceases to be loaded before event 3, for no particular reason except to avoid disturbing Fred's being alive.

Since that paper, Vladimir Lifschitz of McCarthy's group has shown[4] that a new idea, "pointwise circumscription," will solve a simplified version of the Hanks–McDermott problem. No one knows if it solves the more complex version, let alone a realistic set of axioms about physics. No one knows what other problems are still out there. But what's really bothersome about this "solution" is that it is even more top-heavy than previous versions of circumscription. We will have to know the answer, in which case circumscription will verify it for us. In addition, predicates are allowed to be in the class "to-be-minimized" on parts of their domains, and "variable" on other parts, and you have to supply the information about which part is which in the form of an extra relation.

This kind of solution destroys circumscription in order to save it. As with all forms of circumscription, we start with the

[4]See footnote 3.

conclusions that we want to augment our deductive theory with, and we find a second-order axiom that will give us those conclusions. If the first axiom we pick doesn't work, we find a different axiom. Once the exercise is carried out, we throw the axiom away; no one knows how to extract any other consequences than the ones we were verifying. Under these circumstances, what is the axiom doing for us? In what sense is it justifying the conclusions, rather than the desired conclusions justifying it? In practice, it would be just as easy to simply add those conclusions to the theory directly. This procedure would be every bit as nonmonotonic (just change the added ingredients when the theory changes), and every bit as magic.

The original goal, of a simple, general extension of classical logic that would grind out "obviously correct" conclusions, has eluded us. In the case of default formulations, that's because the lures yield nonrecursively enumerable theorems. In the case of circumscription, it's because we have to put the answer in before we can get it out. In both cases, the answers, when available, are often too weak, although with circumscription we often have the option of switching to a different circumscriptive axiom.

It is important to realize that this crisis does not affect programs that reason nonmonotonically. Almost all computerized inference is nonmonotonic and hence nondeductive. That's the problem we started with. What the crisis does affect is our attempt to extend deduction slightly to cover "obvious" cases. As things now stand, there is no nonmonotonic system that justifies the nonmonotonic inferences our programs do. On the contrary, what ends up happening is that we have to expend a lot of effort contorting the formal systems to duplicate simple procedural reasoning. And the effort is a sideshow or afterthought to the development of the program; it doesn't contribute anything.

As I said above, the situation may improve. Someone may discover tomorrow the kind of nonmonotonic system we are looking for. But for now we must conclude that there is no appeal to nonmonotonicity as a way out of some of the problems of deduction.

Doing without deduction

Let us try to summarize the argument so far. I laid out the logicists' project, to express commonsense knowledge in the form of logical axioms. I sketched the justification for their project, and pointed out an implicit premise, that a lot of inference is deductive. I have argued that this premise is wrong, even if logic is extended in various ways.

With this premise knocked out, how does the original argument fare? We can now see that no matter how many axioms you write down about a domain, most of the inferences you want will not follow from them. For that to happen, you must also supply a *program*. In other words, in most cases there is no way to develop a "content theory" without a "process model." (These terms are due to Larry Birnbaum.) A content theory is supposed to be a theory of what people know, how they carve up the world, what their "ontology" is. A process model explains how they use this knowledge. A content theory is at Newell's (1981) "knowledge level," supposedly independent of how the facts it expresses are to be manipulated. What we can now conclude is that content theories are of limited usefulness, in the case where the contemplated inferences are nondeductive. You cannot just start listing facts people know, expressed in logic or any other notation, without saying something about how you assume they will be used by a program, and hence what class of inferences you are trying to account for. The only occasion when you can neglect that chore is when you can point to an important class of purely deductive inferences involving the knowledge. In that case, you do know enough about every candidate process model that you need say no more. But such classes, it now seems, are rare.

By the way, this point should apply just as much to Lenat *et al.*'s (1986) CYC project as to the logicist project. His group has availed themselves of a broader range of tools, and forsworn the discipline of logic, but the same objection presents itself: How will they know when they are making progress?

This argument against free-standing content theories has unfortunate repercussions on the original argument in favor of Tarskian semantics. When there was no program, then denotational semantics was the only way to specify the meanings of our notations. But there is a competing tradition about knowledge representation, which says that a knowledge-representation system is in essence a special-purpose high-level programming language. This point of view is explicit in descriptions of systems like OPS5 (Brownston *et al.* 1985) and Prolog (Clocksin and Mellish 1981), but it applies to many associative nets, too, which are often devices for organizing chunks of Lisp code. Actually, OPS5 and Prolog aren't such great examples, since they are general-purpose programming languages. A better example might be a parser-rule notation like that of Marcus (1980). The notation expresses "knowledge" about the syntax of a language, but it has no denotational semantics. Its semantics are *procedural*; a set of rules is correct if it makes the parser do the right thing.

The competing procedural tradition, in other words, is that a knowledge-representation system does not actually represent anything. This position makes the typical logicist's hair stand on end, because it means acknowledging that the represented knowledge is essentially to be used in just one way. It is hard to count "ways," but picture the "same fact," as needed by two different modules, each with its own special-purpose programming notation. The fact would have to be represented twice. Surely this is not a pleasant requirement to impose on an intelligent program.

It would be nice if a notation could have both denotational and procedural semantics. Nothing prevents this; any logic-based notation that is actually used by a program does *ipso facto* have such a dual semantics. (Pure Prolog programs are an example.) One is tempted to conjecture the converse, that any procedural notation can be translated into an equivalent denotational notation. Isn't it just a matter of cleaning up a few inconsistencies, and making up some ontology? Unfortunately, this optimistic assessment is based on a misconceived notion of how devices like associative networks are actually used. In the minds of some researchers, the notation is supposed to have a formal semantics of some kind, and there is not much doubt that there are equivalent notations that look more like traditional logic. But in the minds of most users, the system is a collection of features — demons and whatnot — just like a standard programming environment (except, they hope, more exotic). Any way of using the features that achieves the immediate programming goal is legitimate. There is nothing shady about this. For every researcher whose system is misused, there are ten who would *encourage* such "creative" use of their system. The chances of being able to find a denotational semantics for any such system are slim.

Still, this deplorable standard of practice cannot by itself deter us from seeking notations that have both denotational and procedural semantics. It's just that this pursuit now seems to lack any rationale. Some people insist that their notations have denotational semantics; others (rather more) can't stand that constraint. In spite of what I am arguing here, I still find myself temperamentally in the first group. If a student comes to me with a denotationless representation, it bothers me. Formerly I thought I had an argument to convince him to rethink, but now all I have is indigestion. The student can always point to his program and claim that it doesn't draw absurd conclusions from his absurd notation. The fact that *I* might draw an absurd conclusion is my problem.

To take one of my favorite examples, consider a simple fact like "The Russians have warships deployed off the US coast." Unless we are willing to resort to "computerdeutsch" predicates like currently_deployed_off_US_coast, a proper representation of something like this will have to express explicitly what the US coast is, roughly how many ships there are and in what distribution they are found, what period of time is implied, and so on and on. But who says that's "proper"? Any particular application program can probably get by with computerdeutsch. And many eager-beaver notation designers will resort to "computerenglisch," such as

(have Russians (deployed warships (off (coast US))))

which seems even worse to me. But why? If the program works, what's wrong with it?

Hence, in the original logicist argument, there is a flaw in the second step, the claim that knowledge must be represented. Although most AI people would assent to this claim, we now see that most of them don't mean it. What they are thinking is roughly: We will have to write a lot of programs to get the knowledge in, and we will need special high-level notations to do it.

The logicist can take comfort in the fact that his opponents have a hard time distinguishing the "high-level" programs that constitute representations from any old programs. If the distinction cannot be made, then all programs could be taken to "represent knowledge," which I take to be the proceduralist position in the old procedural-declarative controversy. This controversy died because no one was really interested in this sense of "represent," by which, for instance, a vision program could be said to represent knowledge about the physics of image formation. There seems to be a stronger sense in which AI programs manipulate explicit representations of objects and facts; denotational semantics provides one answer about what that sense is, but we now see how unattractive this answer is to many AI researchers.

Defending procedures

It's not that the logicist never planned to write programs. He just expected that by the time they were written they would be seen as optimized versions of theorem provers. All that would be required to justify those programs would be to show that they were faithful to the axioms that underlay them.

Now that we have rejected this picture, we need new ways of justifying inferential programs. AI programs are notorious for being impenetrably complex. Sometimes this feature is painted as a virtue, as if the mystery of intelligence ought to be preserved in computational models of it. But a model that we don't understand is not a model at all, especially if it works on only a handful of examples (Marr 1977; Birnbaum 1986).

It is probably impossible to make the idea of "justification" precise enough to support a claim that every program ought to be justified. And yet it is always satisfying when beside a program we can point to a clean, independent theory of why it works. In vision research, for instance, it was a major step just to move from "heterarchical" models, with their air of mystery, to models justified by physics and psychophysics. In the domain of qualitative envisioning (deKleer and Brown 1985; Forbus 1984), there is nothing wrong with the programs that have been written, but it is clarifying to have Kuipers's (1985) analysis of their meaning and limits.

But there are large classes of programs that lack any kind of theoretical underpinnings, especially those concerned with inference to the best explanation, or *abduction*. It would be nice if we could go back to the philosophers and mine their wisdom again. Surely if they could come up with such a great theory of deductive inference they must have done just as well on other kinds, too. Unfortunately, the philosophers have let us down. A theory of abduction might start with answers to questions like these:

What sorts of things need to be explained?
What counts as an explanation?
What counts as evidence for an explanation?
How do you measure the strength of evidential support?
When is evidence strong enough to justify belief in a hypothesis?

So far these questions have received only vague, unmechanizable, piecemeal, or ridiculous answers. We have Bayesian theories, Dempster–Shafer theories, deductive-normological theories, local induction theories, and a lot of arguments about which is best, but none of them answers more than one or two of the questions above, and none seems entirely correct.

This state of affairs does not stop us from writing medical diagnosis programs. But it does keep us from understanding them. There is no independent theory to appeal to that can justify the inferences a program makes. One medical diagnosis program is better than another if fewer of its patients die in clinical trials, I suppose. Actually, what's really bothering me is that these programs embody *tacit* theories of abduction; these theories would be the first nontrivial formal theories of abduction, if only we could make them explicit.

There is an optimistic way and a pessimistic way to view this situation. The pessimistic view is that AI researchers are merely being naive about their chances, buoyed by simple ignorance of the past failures of philosophers. The reason why we cannot extract theories from our programs is that there are no theories to extract. Fodor (1983) puts this conclusion rather grandiloquently at the end of his book *The Modularity of Mind*:

> Localness . . . is a leading characteristic of the sorts of computations that we know to think about. Consider . . . [the] contrast . . . between deductive logic — the history of which is, surely, one of the great success stories of human inquiry — and confirmation theory [i.e., what I was calling abduction theory above] which, by fairly general consensus, is a field that mostly does not exist. My point is that this asymmetry . . . is likely no accident. Deductive logic is the logic of validity, and validity is a *local* property of sentences. . . . The validity of a sentence contrasts starkly with its level of confirmation, since the latter . . . is highly sensitive to global properties of belief systems. . . . We have, to put it bluntly, no computational formalisms that show us how to do this, and we have no idea how such formalisms might be developed. . . . In this respect, cognitive science hasn't even started; we are literally no farther advanced than we were in the darkest days of behaviorism. . . . If someone — a Dreyfus, for

example — were to ask us why we should even suppose that the digital computer is a plausible mechanism for the simulation of global processes, the answering silence would be deafening.

The optimistic view, of course, is that AI researchers can make much faster progress than all those philosophers because we are equipped with "powerful ideas" they didn't have, especially the idea of sophisticated autonomous computation. I hope this is right. But if all we do is go on writing programs, without any general theories emerging, then I am going to get increasingly uncomfortable.

Conclusions

To summarize: The logicist project of expressing "naive physics" in first-order logic has not been very successful. One reason may be that the basic argument was flawed. You cannot write down axioms independent of a program for manipulating them if the inferences you are interested in are not deductions. Unfortunately, very few interesting inferences are deductions, and the attempts by logicists to extend logic to cover more territory have been disappointing. Hence we must resign ourselves to writing programs, and viewing knowledge representations as entities to be manipulated by the programs.

In many respects this is not a critique of logic per se. When you sit down to express a body of knowledge, the notation you use recedes quickly into the background. If you are trying to develop a theory of shape, the constraints imposed by the notational conventions of logic soon dwindle beside the task of trying to express what you know at all. Hence, as I mentioned before, I consider Lenat *et al.*'s (1986) CYC project to be under much the same shadow as the logicists' project.

However, there is one respect in which logic is peculiarly vulnerable, and that is in its resting on denotational semantics. One can accept my conclusions about the futility of formalizing knowledge without a program, and yet still, as I do, have a strong intuition that it is better for a notation to have a denotational semantics than not to. One reason for this might be that at least a sound semantics helps ensure that the deductive inferences done by a program will be right; they may be trivial, but at least they will not be wrong.

Another way of justifying formal semantics has recently been pointed out by Shoham (1986). Suppose a program manipulates a notation, and you can show that the program's conclusions are just those that are true in all A-models of its premises, where what an A-model is depends on the class of inferences you are trying to capture. If the characterization of A-models is intuitively appealing, then you will have provided an independent justification for the operation of the program. If we plug "minimal model" into the schema, we get a program justified, in a way, by circumscription, *except that we dispense with the circumscription axiom*, and just use the semantic notion directly. In the case of temporal inference, the notion of model we need is different; see Shoham (1986) for one proposal. Does this idea apply to a wide variety of types of inference? If so, it provides a way of justifying the ontological and semantic parts of the logicist project, while, alas, dispensing with the idea of programless knowledge representation.

As a tool for studying issues in the semantics and mechanics of knowledge representation, logic still seems unsurpassed. I have in mind examples like Moore's (1980, 1985) work on a computational version of Hintikka's logic of knowledge, which explained how a thinker can refer to unidentified entities whose identities are known by someone else; and Charniak's (1986) work explaining "script variables" as Skolem terms. The insights these papers provide apply to a variety of reasoning programs. Anyone who ignores them just because they are expressed in terms of logic is risking writing an inelegant, irrelevant program.

Finally, I should admit that I am still doing work in the paradigm that I criticize here. In the domain of shape representation, so little is known that focusing on an idealization cannot but help teach us something. The problem I would like to tackle is representing the knowledge required to answer questions like, Could a paper clip be used as a key ring? The idealization I have been forced to fall back on is to prove that a paper clip of a certain size and shape could fit through the hole of a typical key. It should be obvious how much of the original problem this leaves out. Still, the territory is so unexplored that a tour through the idealized fragment could turn up something interesting. What one cannot hope for is to express as logical axioms everything there is to know about using shapes in unusual ways, before designing programs for this task. This will probably come as a shock to no one but me and a few friends.

Acknowledgements

I thank Larry Birnbaum, Steve Hanks, Pat Hayes, Yoav Shoham, and many others for help, some of it against their will. I should also point out that Carl Hewitt, Marvin Minsky, and Bill Woods have been saying similar things for a long time. This work was supported by the National Science Foundation under grant number DCR-8407077.

ALLEN, J. 1984. Towards a general theory of action and time. Artificial Intelligence, **23**(2): 123–154.

BIRNBAUM, L. 1986. Integrated processing in planning and understanding. Yale Computer Science Technical Report 489, Yale University, New Haven, CT.

BROWNSTON, L., FARRELL, R., KANT, E., and MARTIN, N. 1985. Programming expert systems in OPS5: an introduction to rule-based programming. Addison-Wesley Publishing Co., Inc., Reading, MA.

CHARNIAK, E. 1986. Motivation analysis, abductive unification, and nonmonotonic equality. Artificial Intelligence. In press.

CHARNIAK, E., and McDERMOTT, D. 1985. Introduction to artificial intelligence. Addison-Wesley Publishing Co., Inc., Reading, MA.

CLARK, K. L. 1978. Negation as failure. *In* Logic and databases. *Edited by* H. Gallaire and J. Minker. Plenum Press, New York, NY, pp. 293–322.

CLOCKSIN, W., and MELLISH, C. 1981. Programming in Prolog. Springer-Verlag, Berlin, West Germany.

DAVIS, R., and LENAT, D. B. 1982. Knowledge-based systems in artificial intelligence. McGraw-Hill International Book Company, New York, NY.

DEKLEER, J., and BROWN, J. S. 1985. A qualitative physics based on confluences. *In* Formal theories of the commonsense world. *Edited by* J. Hobbs and R. Moore. Ablex Publishing Corporation, Norwood, NJ, pp. 109–183.

DOYLE, J. 1979. A truth maintenance system. Artificial Intelligence, **12**(3): 231–272.

DUDA, R. O., GASCHNIG, J. G., and HART, P. E. 1980. Model design in the Prospector consultant system for mineral exploration. *In* Expert systems in the microelectronic age. *Edited by* D. Michie. Edinburgh University Press, Edinburgh, UK.

FODOR, J. 1983. The Modularity of Mind. MIT Press, Cambridge, MA.

FORBUS, K. 1984. Qualitative process theory. Artificial Intelligence, **24**: 85–168.

GENESERETH, M. R. 1983. An overview of meta-level architecture.

Proceedings of the National Conference on Artificial Intelligence, Washington, DC, pp. 119–124.

HANKS, S., and McDERMOTT, D. 1985. Temporal reasoning and default logics. Computer Science Department Technical Report 430, Yale University, New Haven, CT.

——— 1986. Default reasoning and temporal logics. Proceedings of the National Conference on Artificial Intelligence, Philadelphia, PA.

HAYES, P. J. 1985*a*. The second naive physic manifesto. *In* Formal theories of the commonsense world. *Edited by* J. Hobbs and R. Moore. Ablex Publishing Corporation, Norwood, NJ, pp. 1–20.

——— 1985*b*. Liquids. *In* Formal theories of the commonsense world. *Edited by* J. Hobbs and R. Moore. Ablex Publishing Corporation, Norwood, NJ, pp. 71–107.

HEMPEL, C. G. 1965. Aspects of scientific explanation. Free Press, New York, NY.

HEMPEL, C. G., and OPPENHEIM, P. 1948. Studies in the logic of explanation. Philosophy of Science, **15**: 135–175.

HEWITT, C. 1969. PLANNER: a language for proving theorems in robots. Proceedings of the First International Joint Conference on Artificial Intelligence, Washington, DC, pp. 295–301.

HOBBS, J. 1986. Commonsense summer: final report. Artificial Intelligence Center, SRI International, Menlo Park, CA. Technical note.

HOBBS, J., and MOORE, R. 1985. Formal theories of the commonsense world. Ablex Publishing Corporation, Norwood, NJ.

ISRAEL, D. 1980. What's wrong with non-monotonic logic? Proceedings of the National Conference on Artificial Intelligence, Standford, CA, pp. 99–101.

KUIPERS, B. 1985. The limits of qualitative simulation. Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, CA, pp. 128–136.

LENAT, D. B. 1982. AM: discovery in mathematics as heuristic search. *In* Knowledge-based systems in artificial intelligence. *Edited by* R. Davis and D. B. Lenat. McGraw-Hill International Book Company, New York, NY.

LENAT, D. B., PRAKASH, M., and SHEPHERD, M. 1986. CYC: using commonsense knowledge to overcome brittleness and knowledge acquisition bottlenecks. AI Magazine, **6**(4): 65–85.

LIFSCHITZ, V. 1985. Computing circumscription. Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, CA, pp. 121–127.

LUCKHAM, D. C., and NILSSON, N. J. 1971. Extracting information from resolution proof trees. Artificial Intelligence, **2**(1): 27–54.

MARCUS, M. P. 1980. A theory of syntactic recognition for natural language, MIT Press, Cambridge, MA.

MARR, D. 1977. Artificial Intelligence — a personal view. Artificial Intelligence, **9**: 37–48.

McCARTHY, J. 1980. Circumscription: a nonmonotonic inference rule. Artificial Intelligence, **13**: 27–40.

McDERMOTT, D. 1978. Tarskian semantics or, no notation without denotation! Cognitive Science, **2**(3): 277–282.

——— 1982. A temporal logic for reasoning about processes and plans. Cognitive Science, **6**: 101–155.

——— 1985. Reasoning about plans. *In* Formal theories of the commonsense world. *Edited by* J. Hobbs and R. Moore. pp. 269–317.

McDERMOTT, D., and DOYLE, J. 1980. Non-monotonic logic I. Artificial Intelligence, **13**: 41–72.

MOORE, R. C. 1980. Reasoning about knowledge and action. SRI Artificial Intelligence Center Technical Report 191, SRI International, Menlo Park, CA.

——— 1985. A formal theory of knowledge and action. *In* Formal theories of the commonsense world. *Edited by* J. Hobbs and R. C. Moore. Ablex Publishing Corporation, Norwood, NJ.

NEWELL, A. 1981. The knowledge level. AI Magazine, **1**(3): 1–20.

REITER, R. 1980. A logic for default reasoning. Artificial Intelligence, **13**: 81–132.

ROSENSCHEIN, S. J. 1981. Plan synthesis: a logical perspective. Proceedings of the International Joint Conference on Artificial Intelligence, Vancouver, B.C., pp. 331–337.

SALMON, W. C. 1967. The foundations of scientific inference. University of Pittsburgh Press, Pittsburgh, PA.

——— 1975. Theoretical explanation. *In* Explanation. *Edited by* S. Koerner. Yale University Press, New Haven, CT, pp. 118–145.

SHOHAM, Y. 1985. Naive kinematics: one aspect of shape. Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, CA, pp. 436–442.

——— 1986. Time and causality from the standpoint of artificial intelligence. Ph.D. dissertation, Yale University, New Haven, CT.

SHOHAM, Y., and McDERMOTT, D. 1984. Knowledge inversion. Proceedings of National Conference on Artificial Intelligence, Austin, TX, pp. 295–299.

SHORTLIFFE, E. 1976. Computer-based medical consultations: MYCIN. Elsevier, New York, NY.