REASONING ABOUT KNOWLEDGE AND ACTION

Robert C. Moore
Artificial Intelligence Laboratory
Stanford University
Stanford. California 94S05

Abstract

This paper discusses the problems of representing and reasoning with information about knowledge and action. The first section discusses the importance of having systems that understand the concept of knowledge, and how knowledge is related to action. Section 2 points out some of the special problems that are involved in reasoning about knowledge, and section S presents a logic of knowledge based on the idea of possible worlds. Section 4 integrates this with a logic of actions and gives an example of reasoning in the combined system. Section 5 makes some concluding comments.

1. Introduction

One of the most important concepts an intelligent system needs to understand is the concept of knowledge. AI systems need to understand what knowledge they and the systems or people they interact with have, what knowledge is needed to achieve particular goals, and how that knowledge can be obtained. This paper develops a formalism that provides a framework for stating and solving problems like these. For example, suppose that there is a safe that John wants to open. The common sense inferences that we would like to make might include:

> If John knows the combination, he can immediately open the safe.
>
> If John does not know the combination, he cannot immediately open the safe.
>
> If John knows where the combination is written, he can read the combination and then open the safe.

In thinking about this example, consider how intimately the concept of knowledge is tied up with action. Reasoning about knowledge alone is of limited value. We may want to conclude from the fact that John knows A and B that he must also know C and D, but the real importance of such information is usually that it tells us something about what John can do or is likely to do. A major goal of my research has been to work out some of the interactions of knowing and doing.

That this area has received little attention in AI is somewhat surprising. It is frequently stated that good interactive AI programs will require good models of the people they are communicating with. Surely, one of the most important aspects of a model of another person Is a model of what he knows. The only serious work on these problems in AI which I am aware of is a brief disscussion in McCarthy and Hayes (1969), and some more recent unpublished writings of McCarthy. In philosophy there is a substantial literature on the logic of knowledge and belief. A good introduction to this is Hintikka (1962) and papers by Quine, Kaplan, and Hintikka in Linsky (1971). Many of the ideas I will use come from these papers.

In representing facts about knowledge and actions, I will use first-order predicate calculus, a practice which *is* currently unfashionable. It seems to be widely believed that use of predicate calculus necessarily leads to inefficient reasoning and information retrieval programs. I believe that this is an over-reaction to earlier attempts to build domain-independent theorem provers based on resolution. More recent research, including my own M.S. thesis (Moore, 1975), suggests that predicate calculus can be treated in a more natural manner than resolution and combined with domain-dependent control information for greater efficiency. Furthermore, the problems of reasoning about knowledge seem to require the full ability to handle quantifiers and logical connectives which only predicate calculus posseses.

Section 2 of this paper attempts to bring out some of the special problems involved in reasoning about knowledge. Section 3 presents a formalism which I believe solves these problems, and Section 4 integrates this with a formalism for actions. Section 5 makes some concluding comments.

2. Problems in Reasoning about Knowledge

Reasoning about knowledge presents special difficulties. It turns out that we cannot treat know" as just another relation. If we can represent "Block 1 is on Block2" by 0n(Blockl,Block2), we might be tempted to represent "John knows that P" simply by Know(John,P). This approach glosses over a number of problems. We might be suspicious from the first, since P is not the name of an object but is rather a sentence (or proposition). The semantics of predicate calculus forbid the arbitrary intermingling of sentences and terms for good reason. For one thing, the second argument position of Know is a *referentially opaque context.* Ordinarily in logic we can freely substitute an expression for one that is extensionally equivalent (i.e., one that has the same referent or truth value), without affecting the truth of the formula that contains the expression. This is called *referential transparency.* For example, if X ♦ Y = 7 and X = 3, then 3 ♦ Y = 7. This pattern of reasoning is not valid with Know. We cannot infer from Know(John,(X ♦ Y ■ 7)) and X • 3 that Know(John,(3 ♦ Y ■ 7)) is true, since John might not know the value of X.

One possible solution to this problem is to make the second argument of Know the name of a formula rather than the formula itself. This is essentially the same idea as Goedel numbering, although it is not necessary to use such an obscure encoding as the natural numbers. We won't specify exactly how the encoding is done, but simply use "P" to represent a term denoting the formula P. The representation of "John knows that P" now becomes Know(John."P°). We are no longer in any danger of infering Know(John,^{M}P(A)") from Know(John,"P(BD and A - B, because A is not contained in ^{M}P(A)" Only the name of A, i.e. "A", is contained, and since "A" does not equal "B", there is no problem.

There is, however, a more serious problem, the fact that people can reason with their knowledge. We would expect a reasoning system to have built into it the ability to conclude B from A and A => B But if we treat Know as just an ordinary predicate, we will have no reason to suppose that Know(John,"A") and Know(John,"A B") might suggest KnowUohn.-B"). This problem is emphasised by the fact that there is no formal connection between a formula and its name. The fact that we regard "P" as the name of P is entirely outside the system. To get around this, it is necessary to re-axiomatize the rules of logic within the system, e.g. Va,p,q(Know(a/'p * q") A Know(a,"p") D Know(a,V)). But if we hope to do automated reasoning, this amounts to re-programming the deductive system in first-order logic, and using the top-level inference routines as the interpreter. When we consider the complexities of quantification and matching, it seems likely that this would be an inefficient process.

A different idea which initially seems very appealing is to use the multiple data-base capabilities of advanced AI languages to set up a separate data base for each person whose knowledge we have some information about. We then can record what we know about his knowledge in that data base, and simulate his reasoning by running our standard inference routines in that data base. This idea seems to have wide currency in AI circles, and I advocated it myself in an earlier paper (Moore, 1973).

Unfortunately, it doesn't work very well. It can handle simple statements of the form "John knows that P," but more complicated expressions cause trouble. Consider "John knows that P or John knows that Q." We can't represent this by simply adding 'P or Q," to the data base representing John's knowledge, because this would mean "John knows that P or Q," • something quite different. We could try setting up two data bases, DB1 and DB2, add "P" to one and "Q" to the other, and then assert in the main data base "DB1 represents John's knowledge, or DB2 represents John's knowledge." However, if we also wanted to assert "John knows that C, or John knows that D, or John knows that E," we would need *six* data bases to represent all the possibilites for John's knowledge - one for each of the combinations "A" and "C", B" and "C, A" and "D", etc. As we add more disjunctive assertions, we get a combinatorial explosion in the number of data bases.

We also have a problem in representing "John doesn't know that P." We can't add "not P" to John's data base, because this would be asserting "John knows that not P," and simply omitting "P" from John's data base means that *we* don't know whether John knows that P. So it seems that what John doesn't know has to be kept separate from what he does know. But there are inferences that require looking at both. For example, if we have "John doesn't know that P," and "John knows that Q, implies P," we might want to conclude that "John doesn't know that Q," is probably true. This is representative of a class of inferences that the data base approach doesn't capture. There seems to be a fundamental problem in saying things about a person's knowledge that go beyond simply enumerating what he knows.

3. Reasoning about Knowledge via Possible Worlds

While there may be ways to directly attack the difficulties we have been discussing, there is a way to avoid them entirety by reformulating the problem in terms of possible worlds. When we want to reason about someone's knowledge, rather than talking about what facts he knows, we will talk about which of the various possible worlds might be, so far as he knows, the real world. A person is never completely sure which possible world (or possible state of *the* world) he is in, because his knowledge is incomplete. We will be willing to conclude that a person knows a particular fact, if the fact is true in all the worlds that are possible according to what he knows. This idea is due to Hintikka (1969), and is an adaptation of the semantics for modal logic developed chiefly by Kripke (1963).

Hintikka uses these ideas about possible worlds to provide a model theory for a modal logic of knowledge. In order to use this theory directly for reasoning, we will axiomatize it in first-order logic. To do this, we must encode a language that talks about knowing facts (which we will call the object language) into term expressions of a first-order language that talks about possible worlds (which we will call the meta-language). Then we will have a relation T, such that T(W,P) means the object-language formula denoted by P is true in the possible world denoted by W. So that we can talk more easily about truth in the actual world, we will have a predicate True, such that Trut(P) ■ T(W0,P), where WO Is a constant which refers to the actual world. We will also have a relation K(A,W1,W2), which means that W2 is a world which is possible according to what A knows in Wi. The fundamental axiom of knowledge is then Vwl,a,p(T(wl,Know(a,p) ■ Vw2(K(a,wl,w2) => T(w2,p)). This simply says that a person knows the facts that are true in every world that is possible according to what he knows.

One problem with this axiom is that it is not universally true. For a *perton* to know everything that is true in all worlds which are possible as far as he knows, he would have to know all the logical consequences of his knowledge. Of course, he can know only some of them. But in any particular case, if we can see that a certain conclusion follows from someone's knowledge, we are probably justified in assuming that he can see this also. So we can regard this axiom as a rule of plausible inference, using it when needed, but being prepared to retract our conclusions if they generate contradictions. I will not attempt here to devlop a general theory of plausible reasoning, but I believe that a theory can be worked out that will allow us to use this axiom in essentially its current form.

I should clarify what type of possible worlds I have in mind. Rather than all logically possible worlds, we will consider only those worlds which are possible according to "common knowledge". So, I will feel free to say that facts like "Fish live in water," are true in all possible worlds. This gives us an easy way of saying that not only does everyone know something, but everyone knows that everyone knows it, and everyone knows that everyone knows that everyone knows, etc.

We can now give the full axiomatization of knowledge in terms of possible worlds:

LI.Tru«(pl)«T(WO,pl)
L2. T(wl$_f$(pl And p2)) ■ (T(wl,pl) A T(wl,p2))
L3. T(wl,(pl Or p2)) ■ (T(wl,pl) v T(wl,p2))
L4. T(wl,(pl -> p2)) ■ (T(wl,pl) = T(wl,p2))
L5. T(wl,(pl <•> p2)) ■ (T(wl,pl) ■ T(wl,p2))
L6. T(wl,Not(pl))«-T(wl,pl)

KI. T(wl$_t$Know(al$_f$pl)) • Vw2(K(al,wl,w2) * T(w2,pl))
K2. K(il,wl,wi)
K3. «al,wl,w2) => (K(al,w2,w3) => K(al,wl,w3))
K4. K(il,wl,w2) => (K(al,wl,w3) => K(al,w2,w3))

Axioms LI - L6 just translate the logical connectives from the object language to the meta-language, using the ordinary Tarski definition of truth. For instance, according to L2, (A And B) is true in a world if and only if A is true in the world and B is true in the world. KI is the fundamental axiom of knowledge which we already looked at. K2 says that each world is possible as far as anyone in that world can tell, which is another way of saying that if something is known then it is true. Although it may not be obvious, K3 and K4 imply that everyone knows whether he knows a certain fact. K2 - K4 imply that for fixed A, K(A,wl,w2) is an equivalence relation. This makes our logic of knowledge isomorphic to the modal logic S5. The correspondence between various modal logics and and possible-worlds models for them is discussed in Kripke(1963).

This representation gives us what we need. The meta-language translations of the object-language statements have a structure that reflects their logical properties. To illustrate the use of these axioms, we can prove that people can do simple inferences:

Given True(Know(A,P) And Know(A,(P ■> Q)))

```
Prove: True(Know(A,Q))

 1. True(Know(A,P) And Know(A,(P => Q)))          Given
 2. T(W0,(Know(A,P) And Know(A,(P => Q))))        L1,1
 3. T(W0,Know(A,P)) ∧ T(W0,Know(A,(P => Q)))      L2,2
 4. T(W0,Know(A,P))                               3
 5. K(A,W0,w1) ⊃ T(w1,P)                          K1,4
 6. T(W0,Know(A,(P => Q)))                        3
 7. K(A,W0,w1) ⊃ T(w1,(P => Q))                   K1,6
 8.    K(A,W0,w1)                                 Ass
 9.    T(w1,P)                                    5,8
10.    T(w1,(P => Q))                             7,8
11.    T(w1,P) ⊃ T(w1,Q)                          L4,10
12.    T(w1,Q)                                    11,9
13. K(A,W0,w1) ⊃ T(w1,Q)                          Dis(8,12)
14. T(W0,Know(A,Q))                               K1,13
15. True(Know(A,Q))                               L1,14
```

Proofs in this paper use natural deduction. The right hand column gives the axioms and preceding lines which justify each step. Indented sections are subordinate proofs, and Ass marks the assumptions on which these subordinate proofs are based. Dis indicates the discharge of an assumption.

This proof is completely straight-forward. Lines 1 - 7 simply expand the given facts into possible-worlds notation. Then we pick wl as a typical world which is possible according to what A knows. In lines 9 - 12, we do the inference that we want to attribute to A. Since this inference can be done in an abitrarily chosen member of the set of worlds which are possible for A, it must be valid in all of them (line 13). From this we conclude that A can probably do the inference also (lines 14 - 15).

So far 1 have avoided dealing with the problem of quantifiers. Exactly what do expressions like 3x(Know(A,P(x))). mean? This is not a simple assertion that someone knows a certain fact, so its intuitive meaning may not be clear. The best paraphrase seems to be "There is something that A knows has property P." It is a matter of great dispute in philosophy exactly how to handle this. I will take a pragmatic approach. To say that a person knows of something that it has property P means that he can name something that has property P. Furthermore, just any sort of name won't do. "The thing that has property P is no good, for instance. We will say that A must know the standard name of the thing that has P. This is, of course, a simplification. Not all things have standard names, and some things have different standard names in different contexts, but we will ignore these difficulties to preserve the simplicity of the ordinary case. Abstract entities usually have unproblematical standard names - "23" is the standard name of 23, "15 ♦ 8" is not.

Turning to the model theory, the interpretation of the formula we are considering would be that there is something that is P in all worlds compatible with what A knows. That means that standard names must refer to the same thing in all possible worlds. There is a term for this in philosophy, *rigid designator.* We can greatly simplify our formalism if we require that all ordinary terms in the object language be rigid designators. We would then have to have a special notation for non-rigid designators, but this will not come up in our examples, so I will not develop that idea here. We can now give the axioms for quantifiers and equality:

```
L7. T(w1,Exist(v1,P)) ≡ ∃x(T(w1,P[x/v1]))
      provided x is not free in P

L8. T(w1,All(v1,P)) ≡ ∀x(T(w1,P[x/v1]))
      provided x is not free in P

L9. T(w1,Eq(x1,x2)) ≡ (x1 = x2)
```

L7 and L8 are axiom schemas relative to P and x, and vl is a meta-language variable that ranges over object-language variables. **P[x/v1]** means the result of substituting x for vl in P.

These three axioms may seem somewhat peculiar in that they appear to say that individuals in the world can be part of object-language expressions. In L7 and L8, we took x, a variable ranging over real objects, and inserted it into P, the name of a sentence, implying that objects can be contained in sentences. To preserve the simplicity of the notation, without this apparent absurdity, we will make the interpretation that all functions which represent atomic predicates in the object language (e.g. Eq) take individuals as arguments and return expressions containing the standard names of those individuals.

4. Integrating Knowledge and Action

In order to integrate knowledge with actions, we need to formalize a logic of actions in terms comparable to our logic of knowledge. Happily, the standard AI way of looking at actions does Just that. Most AI programs that reason about actions view the world as a set of possible situations, and each action determines a binary relation on situations, one situation being the outcome of performing the action in the other situation. We will integrate knowledge and action by identifying the possible worlds in our logic of knowledge with the possible situations in our logic of actions.

First, we need to define our formalism for actions exactly parallel to our formalism for knowledge. We will have an object-language relation Res(E,P) which says that it is possible for event E to occur, and P would be true in the resulting situation. In the meta-language, we will have the corresponding relation R(E,W1,W2) which says that W2 is a possible situation/world which could result from event E happening in WI. These two concepts are related in the following way:

```
RI. T(w1,Res(e1,p1)) ≡
       (∃w2(R(e1,w1,w2)) ∧ ∀w2(R(e1,w1,w2) ⊃ T(w2,p1)))
```

The existential clause on the right side of RI says that it is possible for the event to occur, and the universal clause says that in every possible outcome the condition of interest is true. There is a direct parallel here with concepts of program correctness, the first clause expressing termination, and the second, partial correctness.

We can extend the parallel with programming-language semantics to the structure of actions. We will have a type of event which is an actor performing an action, Do(A,C). (C stands for "command".) Actions can be built up from simpler actions using loops, conditionals, and sequences:

```
R2. T(w1,Res(Do(a1,Loop(p1,c1)),p2) ≡
       T(w1,Res(Do(a1,If(p1,(c1;Loop(p1,c1)),Nil)),p2))

R3. T(w1,Res(Do(a1,If(p1,c1,c2)),p2) ≡
       ((T(w1,Know(a1,p1)) ∧ T(w1,Res(Do(a1,c1),p2))) ∨
        (T(w1,Know(a1,Not(p1))) ∧ T(w1,Res(Do(a1,c2),p2))))
R4. T(w1,Res(Do,a1,(c1;c2)),p1)) ≡
      T(w1,Res(Do(a1,c1),Res(Do(a1,c2),p1)))

N1. R(Do(a1,Nil),w1,w2) ≡ (w1 = w2)
```

R2 defines the step-by step expansion of while-loops: if the test is true, execute the body and repeat the loop, else do nothing. To prove general results we would need some sort of induction axiom. R3 defines the execution of a conditional action. Notice that being able to execute a conditional requires *knowing* whether the test condition is true. This differs from ordinary program conditionals, where the test condition is either assumed to be a decidable primitive, or is itself a piece of code to be executed. R4 says that the result of carrying out a sequence of actions is the result of executing the first action, and then executing the rest. NI simply defines the no-op action we need for the definition of Loop.

One of the most important problems we want to look at is how knowledge affects the ability to achieve a goal. Part of the answer is given in the definition of the notion Can. We will say that a person can bring about a condition if and only if there is an action which he knows will achieve the condition:

```
C1. T(w1,Can(a1,p1)) ≡ ∃c1(T(w1,Know(a1,Res(Do(a1,c1),p))))
```

The idea is that to achieve something, a person must know of a plan for achieving it, and then be able to carry out the plan.

We have seen a couple of ways that knowledge affects the possibility of action in R3 and CI. We now want to describe how actions affect knowledge. For actions that are not information-acquiring, we can simply say that the actor knows that he has performed the action. Since our axiomatization of particular actions implies that everyone knows what their effects are, this is sufficient. For information-acquiring actions, like looking at something, we will also add that the information has been acquired. This is best explained by a concrete example. Below, we will work out an example about opening safes, so we will now look at the facts about dialing combinations.

```
D1. ∃w2(R(Do(a1,Dial(x1,x2)),w1,w2) ≡
      (T(w1,Comb(x1)) ∧ T(w1,Safe(x2)) ∧ T(w1,At(a1,x2)))

D2. R(Do(a1,Dial(x1,x2)),w1,w2) ⊃
      ((T(w1,Is-comb-of(x1,x2)) ⊃ T(w2,Open(x2))) ∧
      ((¬T(w1,Is-comb-of(x1,x2)) ∧ ¬T(w1,Open(x2)) ⊃
        ¬T(w2,Open(x2))) ∧
      (T(w1,Open(x2)) ⊃ T(w2,Open(x2)))

D3. R(Do(a1,Dial(x1,x2)),w1,w2) ⊃
      (K(a1,w2,w3) ≡ ((T(w2,Open(x2)) ≡ T(w3,Open(x2))) ∧
        ∃w4(K(a1,w1,w4) ∧ R(Do(a1,Dial(x1,x2)),w4,w3)))
```

DI says that an actor can perform a dialing action if the thing he is dialing is a combination, the thing he is dialing it on is a safe, and he is at the same place as the safe. D2 tells how dialing a combination affects whether the safe is open: if the combination is the combination of the safe, then the safe will be open; if it is not the combination of the safe and the safe was locked, the safe stays locked; if the safe was already open, it stays open.

D3 describes how dialing affects the knowledge of the dialer. Roughly it says that the actor knows he has done the dialing, and he now knows whether the safe *is* open. More precisely, it says that the worlds that are now possible as far as he knows are exactly those which are the result of doing the action in some previously possible world and in which the information acquired matches the actual world. Notice that by making the consequent of D3 a biconditional, we have said that the actor has not acquired any other information by doing the action. Also notice that D3 is more subtle than just saying that whatever he knew before he knows now. This is not strictly true. He might have known before that the safe was locked, and now know that the safe is open. According to D3, if the actor knew before the action "P is true", after the action he knows "P was true before I did this action."

Having presented the basic formalism, I would now like to work out a simple example to illustrate its use. Simply stated, what I will show is that if a person knows the combination of a safe, and he is where the safe is, he can open the safe. Besides the axioms for Dili, we will need two more domain-specific axioms:

```
A1. T(w1,Is-comb-of(x1,x2)) ⊃
      (T(w1,Comb(x1)) ∧ T(w1,Safe(x2)))

A2. T(w1,At(a1,x1)) ⊃ T(w1,Know(a1,At(a1,x1)))
```

A 1 says that if one thing is the combination of another, the first thing is a combination and the second thing is a safe. A2 says that a person knows what is around him. The proof is as follows:

Giv*n: True(At(John,Sf))
Tru«(Exi6ts(X 1 ,Know(John,ls-comb-of (X1 ,Sf))))

Prove: True(Can(John,Open(Sf)))

```
 1. True(Exists(X1,Know(John,Is-comb-of(X1,Sf))))          Given
 2. T(W0,Exists(X1,Know(John,Is-comb-of(X1,Sf))))           L1,1
 3. T(W0,Know(John,Is-comb-of(C,Sf)))                       L7,2
 4. K(John,W0,w1) ⊃ T(w1,Is-comb-of(C,Sf))                  K1,3
 5. True(At(John,Sf))                                       Given
 6. T(W0,At(John,Sf))                                       L1,5
 7. T(W0,Know(John,At(John,Sf)))                            A2,6
 8. K(John,W0,w1) ⊃ T(w1,At(John,Sf))                       K1,7
 9.    K(John,W0,w1)                                        Ass
10.    T(w1,Is-comb-of(C,Sf))                               4,9
11.    T(w1,Comb(C))                                        A1,10
12.    T(w1,Safe(Sf))                                       A1,10
13.    T(w2,At(John,Sf))                                    8,9
14.    ∃w2(R(Do(John,Dial(C,Sf)),w1,w2)                     D1,11,12,13
15.       R(Do(John,Dial(C,Sf)),w1,w2)                      Ass
16.       T(w1,Is-comb-of(C,Sf)) ⊃ T(w2,Open(Sf))           D2,15
17.       T(w2,Open(Sf))                                    16,10
18.    R(Do(John,Dial(C,Sf)),w1,w2) ⊃ T(w2,Open(Sf))        Dis(15,17)
19.    T(w1,Res(Do(John,Dial(C,Sf)),Open(Sf)))              R1,14,18
20. K(John,W0,w1) ⊃                                         Dis(9,19)
       T(w1,Res(Do(John,Dial(C,Sf)),Open(Sf)))
21. T(W0,Know(John,Res(Do(John,Dial(C,Sf)),Open(Sf))))      K1,20
22. T(W0,Can(John,Open(Sf)))                                C1,21
23. True(Can(John,Open(Sf)))                                L1,22
```

The proof is actually simpler than it may look The real work is done in the ten steps between 10 and 19; the other steps are the overhead involved in translating between the object language and the meta language. Notice that we did not have to say explictly that someone needs to know the combination in order to open a safe. Instead we said something more general, that it is necessary to know a procedure in order to do anything. In this case, the combination is part of that procedure. It may also be interesting to point out what would have happened if we had said only that John knew the safe had a combination, but not that he knew what it was. If we had done that, the existential quantifier in the second assertion would have been inside the scope of Know. Then the Skolem constant C would have depended on the variable wl, and the step from 20 to 21 would have failed.

5- Conclusions

In summary, the possible-worlds approach seems to have two major advantages as a tool for reasoning about knowledge. First, it allows "lifting" reasoning in knowledge contexts into the basic deductive system, eliminating the need for separate axioms or rules of inference for these contexts. Second, it permits a very elegant integration of the logic of knowledge with the logic of actions.

This approach seems to work very well as far as we have taken it, but there are some major issues we have not discussed. I have said nothing so far about procedures for reasoning automatically about knowledge. I have some results in this area which appear very promising, but they are too fragmentary for inclusion here. I have also avoided bringing up the frame problem, by not looking at any sequences of action. I am also working in this area, and I consider it one of the largest IOU's generated by this paper. However, the possible-worlds approach has an important advantage here. Whatever method is used to handle the frame problem, whether procedural or axiomatic, knowledge contexts will be handled automatically, simply by applying the method uniformly to all possible worlds. This should eliminate any difficulties of representing what someone knows about the frame problem.

6. References

Hintikka, J. (1963) *Knowledge and Belief.* Ithica, New York: Cornell University Press.

Hintikka, J. (1969) Semantics for Propositional Attitudes, in Linsky (1971), 145-167.

Rripke, S. (1963) Semantical Considerations on Modal Logic, in Linsky (1971), 63-72.

Linsky, L. (ed.) (1971) *Reference and Modality.* London: Oxford University Press.

McCarthy, J. and Hayes, P. J. (1969) Some Philosophical Problems from the Standpoint of Artificial Intelligence, in B. Meltzer and D. Michie (eds.) *Machine Intelligence 4,* 463-502. Edinburgh: Edinburgh University Press.

Moore, R. C (1973) D-SCRIPT: A Computational Theory of Descriptions. *Advance Papers of the Third International Joint Conference on Artificial Intelligence,* 223-229.

Moore, R. C. (1975) *Reasoning from Incomplete Knowledge in a Procedural Deduction System.* MIT Artificial Intelligence Laboratory, AI-TR-347.

A STATE LOGIC FOR, THE PEPPESENTATION OP NATURAL LANGUAGE 12ASED INTELLIGENT SYSTEMS

CamiliaP.Sc)iwind
Tcchnischc Univorsit*~it M'inchen
lunich, FPG

Summary

Tho work described herein introduces a general logic based formalism for the actions of an intelligent system understandinn natural language sentences, executing comnands and answering questions.

natural
language

translation

state logic formulae   derivations ^=-====-   logical axioms

interpretation
of the logic

Kripke-type mod e 1   charnctcrization > :   nnn-logical axioms

The heart of this formal system is a state (or tense) lonic containing special operators for immediately next and preceding states (+,--) as well as for all future states (F) and all past states (P) .

Natural language texts are analysed syntactically and transduced into state logic formulae by an attributed grammar (in the same way as described by Sehwind).

The state logic is formalized by a set of logical axioms and derivation rules for which completeness has been proven. Similar systems have also been mentioned by Rescher. But in usual tense logic systems, the structure of tense has been studied only as to its "pure logical" properties. In intelligent systems however, wo *nood* theorems about the non-logical properties of state changes. The tense structure of a world is determined by changes within the world which affect the non-logical symbols of the world, i.e. the functions or predicates: If a robot takes a block "a" lying on a block "b", then this causes a change of the world (i.e. a state transition) with the meaning of the predicate symbol ON changing. Such non-logical change descriptions are incorporated into our formal system. A model for the state logic is given by a set of classical structures M and a binary relation P on M where s P s' means that the state of the world s immediately precedes the state s^1. Truth values are assigned to formulae depending on the state of the v/orld in which the formula is evaluated. And the state operators take into account the truth value of a formula in some other states which can be "reached" from the actual state. To represent the knowledge incorporated in an intelligent system by such Kripkc-type models we assign a non-logical interpretation to state transitions. The very general model of Kripke-structures is used in such a way that the relation P bears a non-logical meaning. For two structures A_o and A , s P s' holds iff the ''v/orld" ° A , is obtained from the world A„ as the'result of an action which can be executed within A . V^7hat actions can be executed within a v/orld depend on the extensions of the non-logical symbols. On natural language level actions are verbs. The execution of an action has consequences on the extensions of the non-logical symbols of the v/orld, i.e. a structure is subject to some change whenever the action described by the verb is executed in it. If somebody takes a thing the position of that thin^ changes, i.e. the extension of the predicate symbols ON, BEHIND etc. and the extension of the verb predicate symbol HOLD changes, because the person holds the thing now. There are also preconditions for the execution of an action; "a takes $b^{l:}$ is only possible if "a" does not yet hold anything and if "b" has a POsition such that it can be ta*):*on, i.e. there is nothing on "b". We describe both the Preconditions and the consequences of an action by non-logical axioms. And the appropriate structure must have the property that in whatever state all the conditions of an action hold there must be some following state in which its consequences are realized.

Example: Action verb "take"
Precondition axiom (PA)
TAKE x y^HANO x ,-^{m}HING y-\ 1H0LU $_x$ z -> $^{-1}$ ON z y This means: x can take y iff x is a hand and y is a thing and x does not hold any other object and there is nothing on y.
Execution axiom (FA)
TAKE x y-*g+l[HOT,D x y_A~lON y *z]* This means: If x takes y then there is an immediately following state such that x holds y and y is not lying on anything.
We could only describe a small part of the possibilities of our formalism here. Wo actually develop application examples of very different types: one for the analysis of tales and one for traffic.

P.Hayes, A logic of Actions. Machine Intell. 6. pp.495-52o. Ed. B.Meltzcr + D.Michie. Edinb. Univers. Press (1971)
M.Minsky, A Framework for Pepresenting Knowledge. M.I.T.A.I. Memo 3o6 (6.1974)
N.Peschcr + A.Urquhart, Temporal logic. Springer-Verlag, Wien 1971
C.Sehwind, Generating Hierarchical Semantic Networks from Natural Language Discourse. Proceed, of the IJCAI4, (9.1975)
T.Winograd, Understanding Natural Language. Academic Press (1972)

VOCABULARIES FOR PROBLEM SOLVER STATE DESCRIPTIONS

Drew McDermott
Computer Science Department
Yale University
New Haven, Connecticut 06520

ABSTRACT,: A model of knowledge representation is presented and applied to representing problem solvers' states. The focus is on the NASL problem solver, which has been used to study elementary circuit design. The model distinguishes between the form and the content ("vocabulary") of a representation. The vocabularies used by the modules of NASL are displayed. It is concluded that the model allows flexible Implementation and clear description of problem solvers. In particular, it demonstrates the importance of shallow reasoning in the control of problem solvers.

Descriptive terms: Problem solvers, knowledge representation, computer-aided design, production systems, rule-based systems.

Acknowledgements: 1 thank Gerald Sussman for the word "vocabulary," and many of the concepts behind it. Much of the work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense, and monitored by the Office of Naval Research under contract number N00014-75-C-0643.

1 Introduction

This paper is about the "representation of knowledge" in problem-solving systems. 1 focus on the representation of facts about the state of the problem solver itself, rather than on its representations of the current problem state. The discussion uses as an example the NASL problem-solving system. (McDermott, 1976) It is argued that the range of "representational vocabularies" determines the power of a problem solver.

Every representational system has two aspects: form and content. Its form determines what can be inferred from what; its content determines how what is inferred is used by the user of the representational system. In abstract form, such a system can be represented by Fig. 1.

```
Data Base <----> Gatekeeper <----> User Programs
```

Figure 1 General Representation System

The data base is managed by a "gatekeeper." Additions to the data base and requests to the gatekeeper from the user's programs are in terms of the user's concerns. Inside the data base, they are translated into property-list operations, pattern matching, network marking, or some other internal representation for inferential processing.

Ideally, the formal conventions used inside the data base should be hidden from the user. There is a certain level of representational power and retrieval efficiency that he can take for granted (Moore, 1975), so that he will be free to think in problem-oriented terms. Another way to put this is that content must be allowed to be more important than form.

In this model, content appears as the vocabulary the user's programs use in talking to the gatekeeper. Since ray focus in this paper is on the vocabulary, let me expand Fig. 1 to illustrate some points.

```
                                ---> Module 1
                               /
Data Base <---> Gatekeeper <---> Module 2
                               \
                                ---> Module 3
```

Figure 2 Structure of User's Programs

The user's programs often come in "modules," each of which is the primary holder of a specialized vocabulary. For example, in the MYCIN system (Shortliffe, 1976; Davis, 1976), there are two vocabularies: a system of medical terms and a "meta-vocabulary" used in controlling rule application. Inside the data base, most of the rules are in medical terms; the rule-application vocabulary Is used by "meta-rulcs."

An alternative way of approaching these issues would be to let each module have its own data base, gatekeeper, and formal conventions, not just its own vocabulary. For example, in NOAH (Sacerdoti, 1975), the procedural net is stored separately (as special-purpose list structures) from the QL1SP data base used to represent the current problem model. The problem with this approach is that it makes the communication channel between any two modules ad hoc and clumsy. In my model, the data base is the channel; any two rules in the data base can interact, regardless of their origin or, principal vocabulary.

A consequence of my model is that symbols used by user programs have two kinds of "meaning": their "inferential" meaning, determined solely by the behavior of the data-base expressions in which they appear, and their "pragmatic" meaning, which depends on the way in which they alter the behavior of the user programs which employ them. Further, a user module can use a symbol in two different ways: in expressions added to the data base to trigger inferences (thus, as it were, telling itself what it's doing), and in requests to the gatekeeper (which are often asking what to do next). Clearly, the same symbol may be used in both these ways.

II The NASL Problem Solver

NASL is a problem-reduction problem solver which has been applied mainly to the problem of designing simple electronic circuits. (McDermott, 1976) As will be seen shortly, it is implemented as interleaved planning and execution modules. The data base is managed by a PLANNER-like (Hewitt, 1972) predicate-calculus theorem prover, about which I will say no more here. (See McDermott, 1977.) Its "problem vocabulary" includes terms pertaining to devices, circuits, signals, and component values. What I will describe are its "control vocabularies."

A problem solver is a system which solves problems by reducing them to simpler subproblems, and ultimately to "primitives" which can be solved by built-in programs. A problem solver must maintain somewhere a data structure representing important features of its current state. Usually this data structure is different from the data structure representing the current problem state, or "world model." NASL, however, follows the structure of Figure 2, and maintains all its data structures in the same predicate-calculus data base.

This means that the formal system used by NASL restricts as little as possible the content and interaction of the problem-state and problem solver-state descriptions. In what follows, I will first describe the vocabularies used by NASL modules, then show rules which illustrate the interactions of the concepts they define.

NASL looks like this:

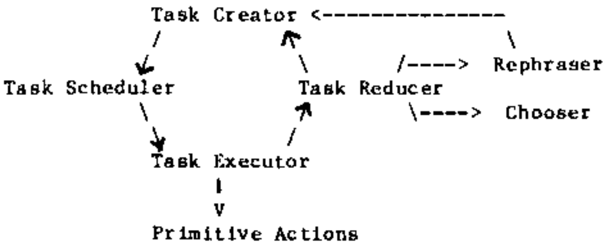

Figure 3 -- NASL Modules

These modules do not correspond one-to-one with programs, but are only conceptual. In Figure 3, an arrow from module A to module B means that A "depends on," "precedes," or "calls" B. The central cycle in the graph is intended to suggest that a problem, or "task," is created, then scheduled, then executed, either as a primitive action or by being reduced to subtasks.

As will be seen shortly, the notations used by these modules result in a task network being represented in the data base.

Two of the modules in Figure 3 are worthy of further description now. The task reducer tries to retrieve exactly one "plan" (set of subtasks) to carry out its task. This attempt can fail in two ways: either too many plans (more than one) are retrieved, or too few (none at all). In the first case, the cholce module tries to choose among them. In the second, the rephrasing module creates a new task which treats the recalcitrant task as an object to be transformed into subtasks; that is, it brings all the problem solver's resources to bear on it.

II.A NASL's Vocabularies and their Pragmatic Meanings

Task Creation and Classification—

The basic predicate for describing tasks Is

```
(TASK name
      < -input-data- > action < -output-data- >)
```

which defines a task with a certain name, which carries out a certain action. Tasks in NASL can deduce or create Information, so there must be some mechanism for passing data from task to task.

For example, this formula

```
(TASK (COUPLER PLAN71)
      <(BUFFER72) (AMP73)>
      (LAMBDA (STAGE1 STAGE2)
         (COUPLE ?STAGE2 ?STAGE1))
      <(CKT74)>)
```

describes a task (COUPLER PLAN71) which couples two circuits and calls the result (CKT74). The action slot must be a function which gives an action given the input data. (Angle brackets surround tuples; "?" indicates a variable.)

I use the neutral word "task" to describe a problem to be worked on, because the concept is intended to be broad. Most problem solvers restrict themselves to actions which cause a (real or simulated) change in the state of the problem model. For example, in the blocks world, actions cause the positions of blocks to change. There are many actions which do not satisfy this definition, for example, "Think of someone who would be willing to lend you $100"; and "While moving block A, avoid disturbing the blocks on top of it." The first example differs from the paradigm in that it causes no change in the world, but only the retrieval of some information. The second is an example of an action ("avoid...") which is executed solely as an influence on the execution of another action ("move..."). Another such "parasitic" action is "Wait here five minutes."

These classifications are indicated by using the predicates

(INFERENTIAL task-name action)

to indicate an information-retrieval task, and

(POLICY task-name action)

to indicate a task which is parasitic.

Task Scheduling—

Tasks are not executed as soon as they are created (unlike the right-hand sides of productions (Newell, 1973)). They may be postponed by rules using the "scheduling vocabulary." The basic predicate here is

(SUCCESSOR task-1 task-2)

which means (roughly) "task-1 must be executed or reduced before task-2." The scheduler examines SUCCESSOR formulas, then sets the. state of a task using the predicate-calculus term

(TASK-STATUS task)

which is successively asserted EQUAL (in the data base) to PENDING, ENABLED, ACTIVE, and FINISHED. (The actual scheme is more complex. See McDermott (1976) for details.)

Task Reduction—

Even when a task is ENABLED, it cannot usually be executed immediately. If its action is not a primitive (i.e., does not have a LISP function to carry it out), it must be <u>reduc</u>ed to subtasks by the task reducer. Such tasks are-called "problematic." The reducer first calls the theorem prover with a request of the form

(TO-DO task action output-data ?PLAN)

to retrieve a plan. A plan is either a single action, or a plan schema ("macro action") which expands into a set of subtasks. (For the details, see. McDermott, 1976.) Either way, new tasks are created by the task reducer, and, for each one, a proposition of the form

(SUBTASK new-task problematic-task)

is recorded.

At any instant, the set of created tasks form a network defined by SUCCESSOR and SUBTASK formulas.

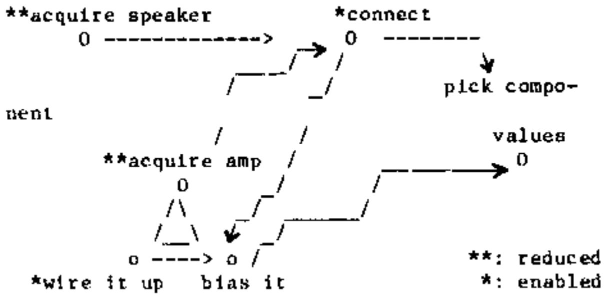

Figure *k* — Task Network Midway through Designing a Hi-Fi

Here the arrows indicate SUCCESSOR formulas. A triangle pointing from low nodes to high indicates SUBTASK formulas; i.e., the low nodes are the sub tasks of the high one.

Primitive Actions—

There are only a few primitive actions in the system. When the user is specifying a new domain, he uses the predicate

(MOD-MANIP task-name action delete-list add-list)

to define new "primitive" actions. (Cf. Fikes and Nilsson, 1971) For example, to define the effect of PUTON in the blocks world, one would write

```
(IMPLIES (ON ?X ?Z)
         (MOD-MANIP ?ANY-TASK (PUTON ?X ?Y)
                    <(ON ?X ?Z)>
                    <(ON ?X ?Y)>))
```

The built-in primitives include

> INFER — for inferring a new fact

> FIND — primitive information retriever

> NO-OP — used for relabeling calculated data

> MONITOR — a primitive policy (used for implementing things like prerequisite protection (Sussman, 1975)) to create an "interrupt" task when its monitored datum is erased.

Choice and Rephrasing—

Finally, there are the vocabularies belonging to the choice and rephrasing modules. The rephraser does not have a very rich vocabulary, a lack which reflects the shallowness of my theories regarding it. Rephrasing currently consists of creating a task with action

(REPHRASE recalcitrant-task action).

This falls in the category of "telling yourself what you are doing." If the user's domain-specific information does not include a plan for rephrasing tasks of this sort, the system will produce a task with action (REPHRASE ... (REPHRASE ...)), for which there is a built-in plan to stop and ask for help.

The choice vocabulary is slightly richer. Rather than merely set up a "choice maker" task, NASL enters a special choice protocol. In this special module, the options (the set of competing plans) are set up by being mentioned in formulas of the form

```
(OPTION choice-name option-name
        option-description)
```

Then a series of requests is made to the information retriever. Each is of one of these forms:

```
(RULE-OUT option-name)
(RULE-IN option-name)
(RULE-TOGETHER < -options- > new-option)
```

Retrieved formulas instruct the choice protocol to eliminate options, favor options, or compose options (in domain-specific ways). The cycle continues until only one option remains, or all options have been ruled out, or no progress has been made on the last loop. For example, the circuit-design knowledge has rules which suggest how to decide between alternative amplifier circuits, and when to try cascading them.

II*B Inferential Meanings of_ Vocabularies

The preceding survey of module vocabularies may be thought of as an informal description of the pragmatic component of their meanings — how they are used by the modules. The richness of the system derives from the "inferential" meanings of the same symbols, determined by their interactions on the other side of the gatekeeper, that is, by the interactions of the rules containing them. These interactions are the medium of communication between modules.

An act of communication occurs when one module adds an expression to the data base from which an expression of interest to another module may be inferred. For example, one rule used by the designer is

```
(IMPLIES
   (POLICY ?P
      (CONSTRAIN 7DEV VOLTAGE-GAIN HIGH))
   (TO-DO ?TSK (MAKE AMPLIFIER) <?DEV>
      (MAKE OP-AMP)))
```

which defines the effect of the parasitic action CONSTRAIN as influencing the execution of MAKE. It suggests MAKE'ing an op-amp as a way of "solving" (MAKE AMPLIFIER) in the case where the amplifier's voltage gain is constrained to be high.

(Before going on, I should point out that not all rules relate two or more module vocabularies. Most TO-DO rules are "plan context" independent. Since in this paper I am emphasizing problem-solver state description, I will ignore the problem of domain-dependent vocabularies and their interactions.)

The most common classes of interactions which 1 have observed are these:

(1) TASK --> SUCCESSOR: Tasks which can interfere with each other must trigger rules to schedule them properly. For example, in electronics, there are rules to the effect that

"Any component-value selection subtask of a design task must follow every topology-altering subtask."

(The full rule, called SELECT-POSTPONE, may be found in McDermott, 1976). Many of the "critics" of Sacerdoti's (1975) NOAH may be thought of as built-in rules of this kind.

(2) POLICY --> TO-DO: Policies are defined as "parasitic" actions which influence other actions. One way this happens is by deduction of TO-DO formulas. 1 gave an example of this above.

(3) POLICY --> CHOICE rule: Another common way to define policies' effects is in terms of their influence on the operation of the choice protocol used to pick among plans. For example, one amplifier-design rule says:

"In choosing between a one-stage common-emitter and a multi-stage, if the bandwidth is CONSTRAINed to be HIGH, RULE-OUT the one-stage."

(Cf. DIFF-CE-N-STAGE, in McDermott, 1976)

(A) TASK —> SUBTASK: Sometimes task reduction occurs entirely via inferential rules; the task reducer just ignores an already-reduced task. For example, bias and coupling plans for multi-stage circuits overlap in their duties. A typical rule from a coupling plan says,

"The tasks for coupling to the second stage also do the work of biasing the base of the second-stage transistor."

(By the way, to make these rules effective, there is another rule, of "interaction type (1)," which says, "Do coupling before biasing." (Cf. Figure 4.) See the rules COUPLE-BEFORE-BIAS and CE-DIR-VOL-COUPLE-PLAN in McDermott, 1976.)

There is nothing special about this list of common interactions; any rule may be used that can be handled by the inferential system. (Of course, we can't expect genius-level insight from the data-base machinery. For instance, we must help it out by telling it whether to use a statement of the form (IMPLIES p q) in a forward or backward direction. Cf. Hewitt, 1972.)

III Results

The NASL system has been applied to two rather different tasks: electronic-circuit design and the blocks world. Both applications are still being debugged. The first domain requires many rules implementing theories of design and electronics; the system currently possesses about 350 rules defining design, elementary electronics, typical circuits, standard biasing plans, and much more; even so, its knowledge is pitifully sketchy compared to what a technician knows. The blocks world was studied for a different reason: as a way of comparing NASL with Sacerdoti's (1975) NOAH; in this domain, the knowledge requires about six pages of rules (on the order of 30 rules).

NASL plus electronics and design rules is called DES1. DESI has never designed a circuit all the way through. It has, however, given ample opportunity for testing the power of NASL'S vocabularies. The system at this writing is capable of doing most of the steps in some simple design tasks. For example, its theory of design specifies a standard "design rephrasing plan" which transforms design problems in ways varying from splitting conjunctive problems into their conjuncts, to translating signal-conversion problems from the time to the frequency domain. (McDermott, 1976, 1978)

The theory of design is fairly domain-specific. That is, there is no general theory of putting to use a new kind of element; instead, there are many prepackaged partial plans for various tasks, and the main job of the designer is to coordinate them. It would be nice if some of the Inferential interaction rules of the last section were deducible from knowledge about new devices, but accomplishing this seems very hard.

After NASL was devised for this task, 1 tried applying it to the rather different blocks world, to see how general it was. My model was the NOAH program, which has several built-in constructs I thought could be expressed as rules. Some could; some couldn't.

For example, the notion of "prerequisite" (Sacerdoti, 1975; Sussman, 1975) is not built into NASL. It can be defined by rules which say

"IF TI's effect P is a prereq of T2, then T2 is a SUCCESSOR of TI, and there is a policy TP to protect P until T2 is begun."

(Protection must be further defined in terms of MONITOR.) The notion of prerequisite can then be used in two ways: passively, to catch "protection violations" (Sussman, 1975); and actively, as a trigger of critics like "Resolve Conflicts." (Sacerdoti, 1975)

One irreducible difference between NASL and NOAH is NASL's inability to reduce a task before its predecessors are reduced. To be able to do this, one must have at least a partial model of the problem state after the predecessors are finished. NOAH can assume such a model is well defined, because NOAH assumes actions are defined as state changes. NASL is pessimistic about the existence of such a model; it never reduces a task until it is time to start executing it.

It is instructive to think of this as a deficiency in vocabulary. That is, there is no action of the form "reduce a task" which NASL can carry out. There is also no built-in term (BEFORE task) which would designate the problem model in which to carry out the reduction (but I believe this could be defined now in terms of things like MOD-MANIP without extensions to NASL). NOAH, on the other hand, has a more limited notion of "task," in which all tasks are non-inferential and non-parasitic, have foreseeable effects, and do not compute results. This enabled its author to separate planning from execution cleanly.

IV Conclusions

1 have presented a model of using a representational system, especially its use by a problem solver to represent its state. The model assigns to each module of a problem solver a vocabulary consisting of symbols with special meanings not derived from inferential interactions. I applied the model to the description of the NASL system (McDermott, 1976), and sketched a comparison with NOAH. (Sacerdoti, 1975)

This model has the following advantages:

(1) Expository clarity — In describing a problem solver, one can factor the description into a formal, representational component, and a "pragmatic" representation user.

(2) Modular implementation — The two pieces can be implemented and optimized separately. (McDermott, 1976)

(3) Ease of comparison — Many differences between programs can be expressed as a difference in vocabulary.

(4) Ease of experimentation — To add a new module is to add a new vocabulary, as far as the representation system is concerned. There are no unnecessary restrictions on form, and no need for a set of special communication channels between a new module and the old ones.

The model brings out these points about NASL

(1) It has an unconstrained notion of problem, the "task." Tasks may be "inferential" or "parasitic," and may compute and receive data from other tasks.

(2) NASL interleaves planning and execution tightly. Planning amounts to the execution of a problematic task.

(3) NASL is incapable of "lookahead" task reduction. There is no built-in term for the state of affairs after an action.

(4) More generally, NASL relies on "shallow reasoning" for thinking about plans. It does not contain a complete axiomatization of any programming language, and cannot prove the correctness of any of its plans. The reason it is successful in spite of these limitations is that it represents its current plan in a very redundant way; almost all features of interest can be retrieved quickly. This means that interesting advice about the domain is likely to be efficiently representable. In a complex and uncertain world, this is probably the best we can do.

The model is still under development. One area in which it needs work is in a general characterization of modules which would allow us to be as unrestrictive yet precise in describing the "pragmatic" component of symbol meaning as predicate calculus allows us to be in describing the "inferential" component. So far, we must make do with English descriptions like those of Section II.

One possibility is to use a production system for this. That Is, a problem solver would be described by a set of "state transition" rules whose left-hand sides described problem-solver states and whose right-hand sides specified changes in terms neutral enough to cover all problem solvers. Modules would correspond to groups of productions sharing a vocabulary. A problem solver would then consist of two complementary pattern-directed systems, one to do inference, the other to do action. This has a certain pleasing symmetry.

References

1. Davis, Randall. Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases. Memo AIM-283, Stanford University Artificial Intelligence Laboratory, 1976.

2. Fikes, R.E. and Nilsson, Nils J. STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 1971, p. 189.

3. Hewitt, Carl. Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot. Technical Report 258, MIT Artificial Intelligence Laboratory, 1972.

4. McDermott, Drew. Flexibility and efficiency in a computer program for designing circuits. Ph.D. thesis, MIT Artificial Intelligence Laboratory, 1976.

5. McDermott, Drew. Deduction in the pejorative sense. Forthcoming, 1977.

6. McDermott, Drew. Circuit design as problem solving. Submitted to IFIPS Working Conference on AI and Pattern Recognition in Computer-Aided Design, 1978.

7. Moore, Robert C. Reasoning from incomplete knowledge in a procedural deductive system. Technical Report 347, MIT Artificial Intelligence Laboratory, 1975.

8. Newell, Allen. Production systems: models of control structures. In Chase, W.C. (Ed.) Visual Information Processing, Academic Press, New York, 1973, p. 463..

9. Sacerdoti, Earl D. A structure for plans and behavior. Technical Note 109, SRI Artificial Intelligence Center, 1975.

10. Shortliffe, Edward H. Computer-Based Medical Consultations: MYCIN, American Elsevier Publishing Company, Inc., New York, 1976.

11. Sussman, Gerald J. *A* Computer Model of Skill Acquisition, American Efsevier Publishing Company, Inc., New York, 1975.