The idea of this theorem is that since it is easier to count than to construct the proofs of complicated theorems, this metatheorem can save you the work of generating a proof. In FOLS metatheory this theorem can be either be proved or simply asserted as axiom.

We use this theorem by directing our attention to the metatheory and instantiating it to some WFF and proving that THEOREM(w). Since we are assuming that our axiomatization of the metatheory is sound, we are then justified in asserting w in the theory. The reflection principles stated below should be looked at as the *reason* that we are justified in asserting w. More detailed examples will be given in the next section.

In FOL we introduce a special LS pair META. It is intended that META is a general theory of LS pairs. When we start, it contains facts about only those things that are common to all LS pairs. Since META behaves like any other first order LS pair additional axioms, etc., can be added to it. This allows a user to assert many other things about a particular theory. Several examples will be given below.

An example of how we axiomatize the notion of well formed

$$\forall Is\ expr.(WFF(expr,Is) \equiv PROPWFF(expr,Is) \vee QUANTWFF(expr,Is))$$

An expression is a WFF (relative to a particular LS pair) if it is either a propositional WFF or a quantifier WFF.

$$\forall Is\ expr.(PROPWFF(expr,Is) \equiv APPLPWFF(expr,Is) \vee AWFF(expr,Is))$$

A propositional WFF is either an application propositional WFF or an atomic WFF.

$$\forall Is\ expr.(APPLPWFF(expr,Is) \equiv PROPCONN(mainsym(expr)) \wedge \forall n.(\emptyset < n \wedge n \leqslant arity(mainsym(expr),Is) \supset WFF(arg(n,expr),Is)))$$

An application propositional WFF is an expression whose main symbol is a propositional connective, and if n is between 0 and the arity of the propositional connective then the n-th argument of the expression must be a WFF. Notice that this definition is mutually recursive with that of PROPWFF and WFF.

$$\forall Is\ expr.(QUANTWFF(expr,Is) \equiv QUANT(mainsym(expr)) \wedge INDVAR(bvar(expr),Is) \wedge WFF(matrix(expr),Is))$$

A quantifier WFF is an expression whose main symbol is a quantifier, whose bound variable is an individual variable and whose matrix is a WFF.

$$\forall Is\ expr.(AWFF(expr,Is) \equiv SENTSYM(expr,Is) \vee APPLAWFF(expr,Is))$$

An atomic WFF is either a sentential symbol or an application atomic WFF.

$$\forall Is\ expr.(APPLAWFF(expr,Is) \equiv PREDSYM(mainsym(expr),Is) \wedge \forall n.(\emptyset < n \wedge n \leqslant arity(mainsym(expr),Is) \supset TERM(arg(n,e),Is)))$$

An atomic application WFF is an expression whose main symbol is a predicate symbol and each argument of this expression in the appropraite range is a TERM.

$$\forall Is\ expr.(TERM(expr,Is) \equiv INDSYM(expr,Is) \vee APPLTERM(expr,Is))$$

$$\forall Is\ expr.(APPLTERM(expr,Is) \equiv OPSYM(mainsym(expr),Is) \wedge \forall n.(\emptyset < n \wedge n \leqslant arity(mainsym(expr,Is)) \supset TERM(arg(n,expr),Is)))$$

A TERM is either an individual symbol or an application TERM, etc.

This is by no means a complete description of LS pairs but it does give some idea of what sentences in META look like. These axioms are collected together in appendix C. The extent of META isn't critical and this paper is not an appropriate place to discuss its details as implemented in FOL. Of course, in addition to the descriptions of the objects contained in the LS pair, it also has axioms about what it means to be a 'theorem', etc.

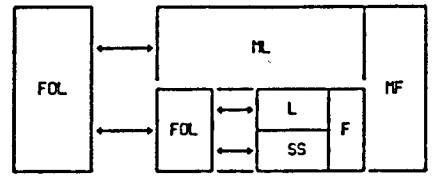

FIG. 2.

Thus META contains the proof theory and some of the model theory of an LS pair. As with any first order theory its language is built up of predicate constants, function symbols and individual constant symbols. What are these? There are constants for WFFs, TERMs, derivations, simulation structures, models, etc. It contains functions for doing 'and introductions', for substituting TERMs into WFFs, constructors and selectors on data structures. It has predicates 'is a well formed formula', 'is a term', 'equality of expressions except for change of bound variables', 'is a model', 'is a simulation structure', 'is a proof', etc.

Suppose that we are considering the metatheory of some particular LS pair, LSO = ⟨L,SS,F⟩. At this point we need to ask a critical question.

What is the *natural* simulation structure for META?

The answer is: (1) we actually have in hand the object we are trying to axiomatize, LSO, and (2) the code of FOL itself contains algorithms for the predicates and functions mentioned above.

This leads to the picture of Fig. 2.It is this picture that leads to the first hint of how to construct a system of logic that can *look* at itself. The trick is that when we carry out the above construction on a computer, the two boxes labeled FOL are physically the same object. I will expand on this in the section on self reflection.

9. Reflection

A *reflection principle* is a statement of a relation between a theory and its metatheory. Although logicians use considerably stronger principles (see Feferman (1962)), we will only use some simple examples, i.e., statements of the soundness of the axiomatization in the metatheory of the theory. An example of a reflection principle is

$$\begin{array}{ll} \text{(in T)} & \backslash\backslash/ \\ & \overline{\quad w \quad} \\ \text{(in MT)} & \text{Prf}(\text{“}\backslash\backslash/\text{”},\text{“}w\text{”}) \end{array}$$

In natural deduction formulations of logic proofs are represented as trees. In the above diagram let '\\/' be a proof and '*w*' be the well formed formula which it proves. Let Prf be a predicate constant in the metatheory, with Prf(p,x) true if and only if p is a proof, x is a wff, and p is a proof of x. Also, let "\\/" and "*w*" be the individual constants in the metatheory that are the names of '\\/' and '*w*', respectively. Then the above reflection principle can be read as: if '\\/' is a proof of '*w*' in the theory we are allowed to assert Prf("\\/","*w*") in the metatheory and vice versa.

A special case of this rule is '*w*' has no dependencies, i.e. it is a theorem.

$$\begin{array}{ll} \text{(in T)} & \overline{\quad w \quad \text{with no dependencies}} \\ \text{(in META)} & \text{THEOREM}(\text{“}w\text{”}) \end{array}$$

A simpler example is

$$\begin{array}{ll} \text{(in T)} & \overline{\text{an individual variable } x} \\ \text{(in META)} & \text{INDVAR}(\text{“}x\text{”}) \end{array}$$

Suppose we have the metatheorem

ANDI: ∀thm1 thm2.THEOREM(mkand(wffof(thm1),wffof(thm2)))

This (meta)theorem says that if we have any two theorems of the theory, then we get a theorem by taking the conjunction of the two wffs associated with these theorems. I need to remark about what I mean by the WFF associated with a theorem. Theorems should be thought of as particular kinds of facts. Facts are more complicated objects than only sentences. They also contain other information. For example, they include the reason we are willing to assert them and what other facts their assertion depends on. Facts also have names. Thus the above line is an incomplete representation of the metatheoretic fact whose name is ANDI. The WFF associated with this fact is just

∀thm1 thm2.THEOREM(mkand(wffof(thm1),wffof(thm2)))

Remember the reflection principle assocaited with THEOREM is

$$\begin{array}{ll} \text{(in T)} & \overline{\quad w \quad \text{with no dependencies}} \\ \text{(in META)} & \text{THEOREM}(\text{“}w\text{”}) \end{array}$$

Thus we can imagine the following scenario.

Suppose we have two theorems called T1 and T2 in the theory. These facts are represented as FOL data structures. Now suppose we want to assert the conjunction of these two in the theory. One way to do this is to use the *and introduction* rule of FOL. This example, however, is going to do it the hard way. First we switch to the metatheory carrying with us the data structures for T1 and T2. We then declare some individual constants t1 and t2 to be of sort theorem in the metatheory, and use the semantic attachment mechanism at the metatheory level to attach the data structures for T1 and T2 to the individual constants t_1 and t_2 respectively. We then instantiate the metatheorem ANDI to t1 and t2. Note that the resulting formula is a ground instance of a sentence without quantifiers. This means that if we have attachments to all the symbols in the formula we can evaluate this formula. In this theorem we have the predicate constant THEOREM. In META it is the only constant in this sentence that is not likely to have an attachment. This is because being a theorem is not in general decidable. Fortunately, we can still use the reflection principle, because we understand the intended interpretation of the metatheory. So if we use the evaluator on

mkand(wffof(t1),wffof(t2)),

we can pick up the data structure computed in the model, instead of the term. Then since we know that it is a theorem we can make it into one in the theory.

This idea has been implemented in a very nice way. In FOL we have the following command.

*****REFLECT ANDI T1,T2;

The reflect command understands some fixed list of reflection principles, which includes those above. When FOL sees the word 'REFLECT' it expects the next thing in the input stream to be the name of a fact in the metatheory of the current theory. So it switches to the metatheory and scans for a fact. It then finds that this fact is universally quantified with two variables ranging over facts in the theory. It switches back to the theory and scans for two facts in the theory. It holds on to the data structures that it gets in that way and switches back to the metatheory. Once there it makes the attachments of these structures to two newly created individual constants, first checking whether or not it already has an attachment for either of these structures. We then instantiate the theorem to the relevant constants and evaluate the result. When we look at the result we notice that it will probably simply evaluate to

THEOREM(mkand(wffof(t1),wffof(t2)))

This is because we don't have an attachment to THEOREM and we also don't have an individual constant which names mkand(wffof(t1),wffof(t2)). But what we do notice is that we have reduced the theorem to the form THEOREM(-), and we know about reflection principles involving THEOREM. Thus we go back and evaluate its argument, mkand(wffof(t1),wffof(t2)), and see if it has a value in the model. In this case since it does we can reflect it back into the theory, by returning to the theory and constructing the appropriate theorem.

This example is a particularly simple one, but the feature described is very general. I will give some more examples below. One thing I want to emphasise here is that what we have done is *to change theorem proving in the theory into evaluation in the metatheory*. I claim that this idea of using reflection with evaluation is the most general case of this and that this feature is not only an extremely useful operation but a fundamental one as well. It is the correct technical realization of how we can *use* declarative information. That is, the only thing you expect a sentence to do is to take its intended interpretation seriously.

The metatheorem we use in reflection does not need to be of the form THEOREM(-). This is the reason for needing the evaluater rather than simply either the syntactic or the semantic simplification mechanisms alone. Consider the following general metatheorems *about* the theory of natural numbers. If you find it hard to read it is explained in detail in the next subsection.

$\forall$vl x.(LINEAREQ(wffof(vl),x) $\supset$ THEOREM(mkequal(x,solve(wffof(vl),x))));
$\forall w\ x$.(LINEAREQ(w,x) $\equiv$
mainsym(w) = Equal $\wedge$
(mainsym(lhs(w)) = Sum $\vee$ mainsym(lhs(w)) = Diff) $\wedge$
larg(lhs(w)) = x $\wedge$
NUMERAL(rarg(lhs(w))) $\wedge$
NUMERAL(rhs(w))) $\wedge$
(mainsym(lhs(w)) = Sum $\supset$ mknum(rhs(w)) < mknum(rarg(lhs(w))));
$\forall w\ x$.(solve(w,x) = IF mainsym(lhs(w)) = Sum
THEN mknumeral(mknum(rhs(w)) − mknum(rarg(lhs(w))))
ELSE mknumeral(mknum(rhs(w)) +
mknum(rarg(lhs(w)))));;

These axioms together with the reflection mechanism extend FOL, so that it can solve equations of the form $x+a = b$ or $x-a = b$, when there is a solution in natural numbers. We could have given a solution in integers or for n simultaneous equations in n unknowns. Each of these requires a different collection of theorems in the metatheory.

This axiomatization may look inefficient but let me point out that solve is exactly the same amount of writing that you would need to write code to solve the same equation. The definition of LINEAREQ is divided into two parts. The first five conjunctions are to do type checking, the sixth conjunct checks for the existence of a solution before you try to use solve to find it. The above example actually does a lot. It type checks the argument, guarantees a solution and then finds it.

9.1. Can a program learn?

In this section I want to digress from the stated intent of the paper and speak a little more generally about AI. It is my feeling that it is the task of AI to explain how it might be possible to build a computer individual that we can interact with as a partner in some problem solving area. This leads to the question of what kinds of conversations we want to have with such an individual and what the nature of our interactions with him should be.

Below I describe a conversation with FOL about solving linear equations. As an example it has two purposes. First it is to illustrate the sense in which FOL is a conversational machine that can have rich discussions (even if not in natural language). And second to explore my ideas of what kinds of dialogues we can have with machines that might be construed as the computer individual learning. I believe that after the discussion presented below we could reasonably say that FOL had learned to solve linear equations. That is, by having this conversation with FOL we have taught FOL some elementary algebra.

Imagine that we have told FOL about Peano arithmetic. We could do this by reading in the axioms presented in Appendix B. We can then have a discussion about *numbers*. For example, we might say

*****ASSUME $n+2 = 7$;
1 $(n+2) = 7$ (1)

and we might want to know what is the value of n. Since we are talking about numbers in the language of Peano arithmetic the *only* way we have of discussing this problem is by using facts about numbers. Suppose that we already know the theorems

THM1: $\forall p\ q\ m.(p = q \supset p-m = q-m)$
THM2: $\forall p\ q\ m.(p+q)-r = p+(q-r)$
THM3: $\forall p.(p+\emptyset) = p$

Then we can prove that $n = 5$

*****$\forall$E THM1 n+2,7,2;
2 $(n+2) = 7 \supset ((n+2)-2) = (7-2)$
*****EVAL BY {THM2,THM3};
3 $(n+2) = 7 \supset n = 5$
***** $\supset$ E 1,3;
4 $n = 5$ (1)

In this case what we have done is proved that $n = 5$ by using facts about *arithmetic*. To put it in the perspective of *conversation*, we are having a discussion about numbers.

If we were expecting to discuss with FOL many such facts, rather than repeating the above conversation many times we might choose to have a single discussion about *algebra*. This would be carried out by introducing the notion of *equation* and a description of how to *solve* them. What is an equation? Well, it simply turns out to be a special kind of *atomic formula* of the theory of arithmetic. That is, we can discuss the solution to equantions by using metatheory.

In FOL we switch to the metatheory. We make some declarations and then define what it means to be a linear equation with a solution by stating the axiom

$\forall w\ x.$(LINEAREQ(w,x) ≡
mainsym(w) = Equal ∧
(mainsym(lhs(w)) = Sum ∨ mainsym(lhs(w)) = Diff) ∧
larg(lhs(w)) = x ∧
NUMERAL(rarg(lhs(w))) ∧
NUMERAL(rhs(w)) ∧
(mainsym(lhs(w)) = Sum ⊃ mknum(rhs(w)) > mknum(rarg(lhs(w)))))

Here w is a (meta)variable ranging over WFFs, and x is a (meta)variable ranging over individual variables. Spelled out in English this sentence says that a well formed formula is a linear equation if and only if:

(i) it is an equality,

(ii) its left hand side is either a sum or a difference,

(iii) the left hand argument of the left hand side of the equality is x,

(iv) then right hand argument of the left hand side of the equality is a numeral,

(v) the right hand side of the equality is a numeral and

(vi) if the left hand side is a sum then the number denoted by the numeral on the right hand side is greater than the number denoted by the numeral appearing in the left hand side.

In more mathematical terminology it is: that the well formed formula must be either of the form $x+a = b$ or $x-a = b$ where a and b are numerals and x is an individual variable. Since here we are only interested in the natural numbers, the last restriction in the definition of LINEAREQ is needed to guarantee the existence of a solution.

We also describe how to find out what is the *solution* to an equation.

$\forall w\ x.$(solve(w,x) = IF mainsym(lhs(w)) = Sum
THEN mknumeral(mknum(rhs(w))−
mkum(rarg(lhs(w))))
ELSE mknumeral(mknum(rhs(w))+
mknum(rarg(lhs(w)))));;

This is a function definition in the meta theory. Finally we assert that if we have an equation in the theory then the numeral constructed by the solver can be asserted to be the answer.

$\forall$vl $x.$(LINEAREQ(wffof(vl),x) ⊃ THEOREM(mkequal(x,solve(wffof(vl),x)))));

We then tell FOL to remember these facts in a way that is convenient to be used by FOL's evaluator.

This then is the conversation we have with FOL about equations. Now we are ready to see how FOL can use that information, so we switch FOL's attention back to the theory. Now, whenever we want to solve a linear equation, we simply remark, using the reflect command, that he should remember our discussion about solving equations.

We can now get the effect of the small proof above by saying

*****REFLECT SOLVE 1;
$5n = 5$ (1)

In effect FOL has learned to solve simple linear equations.

We could go on to ask FOL to prove that the function solve actually provides a solution to the equation, rather than our just telling FOL that it does, but this is simply a matter of sophistication. It has to do with the question of what you are willing to accept as a justification.

One reasonable justification is that the teacher told me. This is exactly the state we are in above. On the other hand if that is not satisfactory then it is possible to discuss with FOL the justification of the solution. This could be accomplished by explaining to FOL (in the metatheory) not to assert the solution of the equations in the theory, but rather to construct a proof of the correctness of the solution as we did when we started. Clearly this can be done using same machinery that was used here. This is important because it means that our reasoning system does not need to be expanded. We only have to tell it more information.

A much more reasonable alternative is to tell FOL (again in the metatheory) two things. One is what we have above, i.e., to assert the solution of the equation. Second is that if asked to justify the solution, then to produce that proof. This combines the advantages each of the above possibilities. I want to point out that this is very close to the kinds of discussions that you want to be able to have with people about simple algebra.

Informally we always speak about solving *equations*. That is, we think of them as syntactic and learn how to manipulate them. This is not thinking about them as relations, which is their usual first order interpretation. In this sense going to the metatheory and treating them as syntactic objects is very close to our informal use of these notions.

I believe that this is exactly what we want in an AI system dealing with the question of *teaching*. Notice that we have the best of both worlds. On the one

hand, at the theory level, we can 'execute' this learning, i.e. use it, and on the other hand, at the metatheory level, we can reason about what we have learned about manipulating equations. In addition the correct distinction between equations as facts and equations as syntactic objects has been maintained. The division between theory and metatheory has allowed us to view the same object in both these ways without contradiction or the possibility of confusion.

As is evident from the above description, one of the things we have here is a very general purpose programming system. In addition it is extendable. Above we have showed how to introduce any new subsidiary deduction rule that you chose, 'simply' by telling FOL what you would like it to do. This satisfies the desires of Davis and Schwartz (1977) but in a setting not restricted to the theory of hereditarily finite sets. As I said above: we are using first order logic in what I believe is its most general and natural setting.

There are hundreds of examples of this kind where their natural description is in the metatheory. In a later paper I will discuss just how much of the intent of natural language can only be understood if you realize that a lot of what we say is about our use of language, not about objects in the world. This kind of conversation is most naturally carried out in the metatheory with the use of the kind of self-reflective structures hinted about below.

9.2. Using metametatheory

We can take another leap by allowing ourselves to iterate the above procedure and using metametatheory. This section is quite sketchy but would require a full paper to write out the details.

We can use the metametatheory to describe declaratively what we generally call heuristics. Consider an idealized version of the Boyer and Moore (1979) theorem prover for recursive functions. This prover looks at a function definition and tries to decide whether or not to try to prove some property of the function using either CAR-induction or CDR-induction, depending on the form of the function definition.

CAR and CDR inductions are axiom schemas, which depend on the form of the function definition and the WFF being proved. Imagine that these had already told to FOL in the metatheory. Suppose we had called them CARIND and CDRIND. Then using the facilities described above we could use these facts by reflection. For example,

*****ASSUME $\forall u.\text{counta}(u) = \text{if atom}(u) \text{ then } u \text{ else counta}(\text{car}(u))$;

1 $\forall u.\text{counta}(u) = \text{if atom}(u) \text{ then } u \text{ else counta}(\text{car}(u))$ (1)

*****REFLECT CARIND 1 $\forall u.\text{ATOM}(\text{counta}(u))$;

2 $\forall u.\text{ATOM}(\text{counta}(u))$ (1)

*****ASSUME $\forall u.\text{countd}(u) = \text{if null}(u) \text{ then 'NIL else countd}(\text{cdr}(u))$;

3 $\forall u.\text{countd}(u) = \text{if null}(u) \text{ then 'NIL else countd}(\text{cdr}(u))$ (3)

*****REFLECT CDRIND 3 $\forall u.\text{countd}(u) = \text{'NIL}$;

4 $\forall u.\text{countd}(u) = \text{'NIL}$ (3)

The use of this kind of command can be discussed in the metametatheory. We introduce a function, T_reflect, in the metametatheory, which we attach using semantic attachment to the FOL code that implements the above reflect command. Thus T_reflect takes a fact, $\vee$I and a list of arguments, and if it succeeds returns a new proof whose last step is the newly asserted fact and if it fails returns some error. Suppose also that Carind and Cdrind are the metametatheory's name for CARIND and CDRIND respectively. Then suppose in the metametatheory we let

```
WFF1 = mkforall(T_u,mkapplw1(T_ATOM,
                             (mkappl1(T_counta,T_u))))
WFF2 = mkforall(T_u,mkequal(mkappl1(T_countd,T_u),
                            mksexp('NIL)))
```

that is, $Au.\text{ATOM}(\text{counta}(u))$ and $\forall u.\text{countd}(u) = \text{'NIL}$, respectively. We prefix things refering to the theory by "T_". The effect of the above commands (without actually asserting anything) is gotten by using the FOL evaluator on

```
T_reflect(Cdrind, < T_fact(1),WFF1 > )
T_reflect(Cdrind, < T_fact(3),WFF1 > ).
```

Now suppose that $\vee$I ranges over facts in the theory, f ranges over function symbols, and w ranges over WFFs. The micro Boyer and Moore theorem prover can be expressed by

$$
\begin{aligned}
&\forall \vee \mathrm{I}\, f\, w.\\
&\quad (\text{IS\_T\_FUNDEF}(\vee \mathrm{I}, f) \supset\\
&\qquad (\text{CONTAINS\_ONLY\_CAR\_RECURSION}(\vee \mathrm{I}, f) \wedge\\
&\qquad\quad \text{NOERROR}(\text{T\_REFLECT}(\text{Carind}, \langle \vee \mathrm{I}, w \rangle)) \supset\\
&\qquad\quad \text{T\_THEOREM}(\text{last\_T\_step}(\text{T\_REFLECT}(\text{Carind}, \langle \vee \mathrm{I}, w \rangle)))) \wedge\\
&\qquad (\text{CONTAINS\_ONLY\_CDR\_RECURSION}(\vee \mathrm{I}, f) \wedge\\
&\qquad\quad \text{NOERROR}(\text{T\_REFLECT}(\text{Cdrind}, \langle \vee \mathrm{I}, w \rangle)) \supset\\
&\qquad\quad \text{T\_THEOREM}(\text{last\_T\_step}(\text{T\_REFLECT}(\text{Cdrind}, \langle \vee \mathrm{I}, w \rangle)))))
\end{aligned}
$$

In the metametatheory we call this fact BOYER_and_MOORE. It is read as follows: if in the theory, $\vee$I is a function definition of the function symbol f, then if this function definition only contains recursions on car, and if when you apply reflection from the theory level to the metatheorem called Carind you don't get an error, then the result of this reflection is a theorem at the theory level, similarly for cdr induction.

As explained in the previous sections, asserting this in the metametatheory allows it to be used at the theory level by using the same reflection device as before.

When our attention is directed to the theory we can say

*****MREFLECT BOYER_and_MOORE 1,counta,$\forall u$.ATOM(counta(u));

5 $\forall u$.ATOM(counta(u)) (1)

*****MREFLECT BOYER_and_MOORE 3,countd,$\forall u$.countd(u) = 'NIL;

6 $\forall u$.countd(u) = 'NIL (3)

Here MREFLECT simply means reflect into the metametatheory.

This example shows how the metametatheory, together with reflection, can be used to drive the proof checker itself. Thus we have the ability to declaratively state heuristics and have them effectively used. The ability to reason about heuristics and prove additional theorems about them provides us with an enormous extra power. Notice that we have once again changed theorem proving at the theory level into computation at the metametatheory level. This is part of the leverage that we get by having all of this machinery around simultaneously.

This example, as it is described above, has not yet been run in FOL. It is the only example in this paper which has not actually been done using the FOL system, but it is clear that it will work simply given the current features.

A good way of looking at all of this is that the same *kind* of language that we use to carry on ordinary conversations with FOL can be used to discuss the control structures of FOL itself. Thus it can be used to discuss its own actions.

10. Self Reflection

In the traditional view of metatheory we start with a theory and we axiomatize that theory. This gives us metatheory. Later we may axiomatize that theory. That gives us metametatheory. If you believe that most reasoning is at some level (as I do) then this view of towers of metatheories leads to many questions. For example, how is it that human memory space doesn't overflow. Each theory in the tower seems to contain a complete description of the theory below thus exponentiating the amount of space needed!

In the section on metatheory, I introduced the LS pair, META. Since it is a first order theory just like any other, FOL can deal with it just like any other. Since META is the general theory of LS pairs and META is an LS pair this might suggest that META is also a theory that contains facts about itself. That is, by introducing the individual constant Meta into the theory META and by using semantic attachment to attach the theory (i.e., the actual machine data structure) META to Meta we can give META its own name. The rest of this section is somewhat vague. We have just begun to work out the consequences of this observation.

FOL handles many LS pairs simultaneously. I have showed how given any LS pair we can direct FOL's attention to META using reflection. Once META has an individual constant which is a name for itself and we have attached META to this constant, then META is FOL's theory of itself. Notice several things:

(1) If META has names for all of the LS pairs known to FOL then it has the entire FOL system as its simulation structure;

(2) Since META is a theory about any LS pair, we can use it to reason about itself.

We can illustrate this in FOL by switching to META and executing the following command.

*****REFLECT ANDI ANDI ANDI;

1 $\forall$thm1 thm2.THEOREM(mkand(wffof(thm1),wffof(thm2))) $\wedge$
$\forall$thm1 thm2.THEOREM(mkand(wffof(thm1),wffof(thm2)))

The effect we have achieved is that when FOL's attention is directed at itself, then when we reflect into its own metatheory we have a system that is capable of reasoning about itself.

When looking at human reasoning I am struck by two facts. First, we seem to be able to apply the Meta facts that we know to any problems that we are trying to solve, and second, even though it is possible to construct simple examples of use/mention conflicts, most people arrive at correct answers to questions without even knowing there is a problem. Namely, although natural language is filled with apparent puns that arise out of use/mention confusions, the people speaking do not confuse the names of things with the things. That is, the *meaning* is clear to them.

The above command suggests one possible technical way in which both of these problems can be addressed. The structure of FOL *knew* that the first occurrence of ANDI in the above command was a 'use' and that the second and third were 'mentions'. Furthermore, the same routines that dealt effectively with the ordinary non self reflective way of looking at theory/metatheory relations also dealt with this case of self reflection without difficulty.

Notice that I said, 'this case'. It is possible with the structure that I have described above to ask META embarrassing questions. For example, if you ask META twice in a row what the largest step number in its proof is you will get two different answers. This would seem to lead to a contradiction.

The source of this problem is in what I believe is in our traditional idea of what it means to be a *rule of inference*. Self reflective systems have properties that are different from ordinary systems. In particular, whenever you 'apply a rule of inference' to the facts of this system you change the structure of META itself and as a result you change the attachment to Meta. This process of having a rule of inference changes the models of a theory as well as the already proven facts simply does not happen in traditional logics. This change of point of view requires a new idea of what is a valid rule of inference for such systems.

The extent of the soundness of the structure that I propose here is well beyond the scope of this elementary paper. Also FOL was built largely before I understood anything about this more general idea of rule of inference, thus the current FOL

code cannot adequately implement these ideas. One of many main current research interests is in working out the consequences of these self reflective structures.

META has many strange properties which I have just begun to appreciate and is a large topic for further research.

11. Conclusion

11.1. Summary of important results

I want to review what I consider to be the important results of this paper.

One is the observation that, when we reason, we use representations of the objects we are reasoning about as well as a representation of the facts about these objects. This is technically realized by FOL's manipulation of LS pairs using semantic attachment. It is incorrect to view this as a procedural representation of facts. Instead we should look at it as an ability to explicitly represent procedures. That is, simulation structures give us an opportunity to have a machine representation of the objects we want to reason about as well as the sentences we use to mention them.

Second, the evaluator I described above is an important object. When used by itself it represents a mathematical way of describing algorithms together with the assurance that they are correctly implemented. This is a consequence of the fact that the evaluator only performs logically valid transformations on the function definitions. In this way we could use the evaluator to actually generate a proof that the computed answer is correct. In these cases evaluation and deduction become the same thing. This is similar in spirit to the work of Kowalski (1974), but does not rely on any normalization of formulas. It considers the usual logical function definitions and takes their intended interpretation seriously. This evaluator works on any expression with respect to any LS pair and its implementation has proved to be only two to three times slower than a lisp interpreter.

Third is the observation that the FOL proof checker is itself the natural simulation structure for the theory META of LS pairs. This gives us a clean way of saying what the intended interpretation of META is. This observation makes evaluation in META a very powerful tool. It is also the seed of a theory of self reflective logic structures that, like humans, can reason about themselves.

Fourth is the use of reflection principles to connect an LS pair with META. This, together with the REFLECT command, is a technical explanation of what has been called the declarative/procedural controversy. Consider the META theorem ANDI described above. When we use the REFLECT command to point at it from some LS pair, ANDI is viewed procedurally. We want it to do an *and introduction*. On the other hand when we are reasoning in META, it is a sentence like any other. Whether a sentence is looked at declaratively or procedurally depends on your point of view, that is, it depends where you are standing when you look at it.

I have presented here a general description of a working reasoning system that includes not only theories but also metatheories of arbitrarily high level. I have given several examples of how these features, together with reflection can be used to dynamically extend the reasoning power of the working FOL system. I have made some references to the way in which one can use the self reflective parts of this system. I have given examples of how heuristics for using subsidiary deduction rules can be described using these structures. In addition, since everything you type to FOL refers to some LS pair, all of the above things can be reasoned about using the same machinery.

11.2. Concluding remarks, history and thanks

I have tried in this paper to give a summary of the ideas which motivate the current FOL system. Unfortunately this leaves little room for complex examples so I should say a little about history, the kinds of things that have been done and what is being done now.

FOL was started in 1972 and the basic ideas for this system were already known in the summer of 1973. Many of the ideas of this system come directly out of taking seriously John McCarthy's idea that before we can ever expect to do interesting problem solving we need a device that can represent the ideas involved in the problem. I started by attempting to use ordinary first order logic and set theory to represent the ideas of mathematics. My discovery of the explicit use of computable partial models (i.e. simulation structures) came out of thinking about a general form for what McCarthy (1973) called a 'computation rule' for logic, together with thinking about problems like the one about real numbers mentioned above. The first implementation of semantic evaluation was in 1974 by me. Since then it has been worked on by Arthur Thomas, Chris Goad, Juan Bulnes and most recently by Andrew Robinson. The first aggressive use of semantic attachment was by Bob Filman (1978) in his thesis.

This idea of attaching algorithms to function and predicate letters is not new to AI. It appears first in Green (1969) I believe, but since then in too many places to cite them all. What is new here is that we have done it uniformly, in such a way that the process can be reasoned about. We have also arranged it so that there can be no confusion between what parts of our data structure is code and what parts are sentences of logic.

The real push for metatheory came from several directions. One was the realization that most of mathematical reasoning in practice was metatheoretic. This conflicted with most current theorem proving ideas of carrying out the reasoning in the theory itself. Second was my desire to be able to do the proofs in Kleene (1952) about the correctness of programs. In the near future we are planning to carry out this dream. Carolyn Talcott and I plan to completely formalize LISP using all the power of the FOL system described above. In addition there will be people working on program transformations in the style of Burstall and Darlington

(1977). The third push for metatheory was a desire to address the question of common sense reasoning. This more than anything needs the ability to be able to reason about our theories of the world. One step in this direction has been taken by Carolyn Talcott and myself. We have worked out D. Michie's Keys and boxes problem using this way of thinking and are currently writing it up.

The desire to deal with metatheory led to the invention of the FOL reflection command. Metatheory is pretty useless without a way of connecting it to the theory. I believe that I am the first to use reflection in this way.

All of the above ideas were presented at the informal session at IJCAI 1973. This panel was composed of Carl Hewitt, Allen Newell, Alan Kay and myself.

The idea of self reflection grew out of thinking about the picture in the section on metatheory.

It has taken several years to make these routines all work together. They first all worked in June 1977 when Dan Blom finished the coding of the evaluator. I gave some informal demos of the examples in this paper at IJCAI 1977.

I suppose that here is as good a place as any to thank all the people that helped this effort. Particularly John McCarthy for his vision and for supporting FOL all these years. I would not have had as much fun doing it alone. Thanks.

I hope to write detailed papers on each of these features with substantial examples. In the meantime I hope this gives a reasonable idea of how FOL works.

Appendices

A. An axiomatization of natural numbers

The commands below repeat those given in Section 4. They will be used in the examples below. One should keep in mind that this is an axiomatization of the natural numbers (including 0), not an axiomatization of the integers.

```
DECLARE INDVAR n m p q ∈ NATNUM;
DECLARE OPCONST suc pred(NATNUM) → NATNUM;
DECLARE OPCONST +(NATNUM, NATNUM) → NATNUM [R→ 458,L→ 455];
DECLARE OPCONST *(NATNUM, NATNUM) → NATNUM [R→ 558,L→ 5551;
DECLARE PREDCONST < (NATNUM, NATNUM) [INF];
DECLARE PREDPAR P(NATNUM);
AXIOM  Q:
       ONEONE: ∀n m.(suc(n) = suc(m) ⊃ n = m);
       SUCC1:  ∀n.¬ (0 = suc(n));
       SUCC2:  ∀n.(¬ 0 = n ⊃ ∃m.(n = suc(m)));
       PLUS:   ∀n.n+0 = n
               ∀n m.n+suc(m) = suc(n+m);
       TIMES:  ∀n.n*0 = 0
               ∀n m.n*suc(m) = (n*m)+m; ;;
AXIOM  INDUCT: P(0)∧ ∀n.(P(n) ⊃ P(suc(n))) ⊃ ∀n.P(n); ;
REPRESENT {NATNUM} AS NATNUMREP;
ATTACH suc  ↔ (LAMBDA (X) (ADD1 X));
ATTACH pred ↔ (LAMBDA (X) (COND ((GREATERP X 0) (SUB1 X)) (T 0)));
ATTACH+     ↔ (LAMBDA (X Y) (PLUS X Y));
ATTACH*     ↔ (LAMBDA (X Y) (TIMES X Y));
ATTACH <    ↔ (LAMBDA (X Y) (LESSP X Y));
```

B. An axiomatization of s-expressions

These commands describe to FOL a simple theory of s-expressions. In addition it contains the definitions on the functions @, for appending two lists, and rev, for reversing a list.

```
DECLARE INDVAR x y z ∈ Sexp;
DECLARE INDVAR u v w ∈ List;
DECLARE INDCONST nil ∈ Null;

DECLARE OPCONST car cdr 1;
DECLARE OPCONST cons(Sexp,List) → List;
DECLARE OPCONST rev 1;
DECLARE OPCONST @ 2 [inf];

DECLARE SIMPSET Basic;
DECLARE SIMPSET Funs;

MOREGENERAL Sexp ⊇ {List, Atom, Null};
MOREGENERAL List ⊇ {Null};

REPRESENT {Sexp} AS SEXPREP;

AXIOM CAR: ∀x y.car(cons(x,y)) = x;;
AXIOM CDR: ∀x y.cdr(cons(x,y)) = y;;
AXIOM CONS: ∀x y.¬ Null(cons(x,y));;

Basic← {CAR,COR,CONS};

AXIOM REV: ∀u.(rev(u) = IF Null (u) THEN u ELSE rev(cdr(u)) @ cons(car(u),nll));;
AXIOM APPEND: ∀u v.(u@v = IF Null(u) THEN  v ELSE cons(car(u),cdr(u)@v);;

Funs← {REV,APPEND};
```

C. An axiomatization of well formed formulas

This is an example of how WFFs are axiomatized in META. It simply collects together the formulas of Section 8

```
∀Is expr.(WFF(expr,Is) ≡ PROPWFF(expr,Is) ∨
                                    QUANTWFF(expr,Is))
∀Is expr.(PROPWFF(expr,Is) ≡ APPLPWFF(expr,Is) ∨
                                                AWFF(expr,Is))
∀Is expr.(APPLPWFF(expr,Is) ≡ PROPCONN(mainsym(expr)) ∧
      ∀n.(0 < n ∧ n ⩽ arity(mainsym(expr),Is) ⊃
                                                WFF(arg(n,expr),Is)))
∀Is expr.(QUANTWFF(expr,Is) ≡
        QUANT(mainsym(expr)) ∧ INDVAR(bvar(expr),Is ∧
                                                WFF(matrix(expr),Is)
∀Is expr.(AWFF(expr,Is) ≡ SENTSYM(expr,Is) ∨
                                    APPLAWFF(expr,Is)
∀Is expr.(APPLAWFF(expr,Is) ≡ PREDSYM(mainsym(expr),Is) ∧
      ∀n.(0 < n ∧ n ⩽ arity(mainsym(expr),Is) ⊃
                                    TERM(arg(n,e),Is)))
∀Is expr.(TERM(expr,Is) ≡ INDSYM(expr,Is) ∨ APPLTERM(expr,Is))
∀Is expr.(APPLTERM(expr,Is) ≡ OPSYM(mainsym(expr),Is) ∧
      ∀n.(0 < n ∧ n ⩽ arity(mainsym(expr,Is)) ⊃
                                    TERM(arg(n.expr),Is)))
```

D. Examples of semantic evaluations

We give two sets of examples of semantic evaluation.

In the theory of s-expressions

```
*****DECLARE OPCONST length(Sexp) = Sexp;
*****ATTACH length ↔ LENGTH;
length attached to LENGTH
*****SIMPLIFY length('(A B));
1 length ('(A B)) = '2
*****SIMPLIFY length ('(A B)) = 2;
2 length ('(A B)) = 2 ≡ '2 = 2
*****SIMPLIFY '2 = 2;
Can't simplify
*****SIMPLIFY '2 = '4;
3¬ ('2 = '4)
```

In the theory of natural numbers

```
%6*****SIMPLIFY 2+3 < pred(7);
1 2+3 < pred(7)
*****SIMPLIFY 4*suc(2)+pred(3) < pred(pred(8));
2¬ 4*suc(2)+pred(3) < pred(pred(8))
*****SIMPLIFY n*0 < 3;
no simplifications
```

E. An example of syntactic simplification

After

```
*****simplify Null(nil);
1 Null(nil)
```

the command

REWRITE rev cons(x,nil) BY Basic ∪ Funs ∪ {1} ∪ LOGICTREE;

produces the result

2 rev(cons(x,nil)) = cons(x,nil)

by a single syntactic simplification. The exact details of what the simplifier did are recorded below. The numbers on the left refer to notes below the example.

```
Trying to simplify
|     rev(cons(x,nil))
|
succeeded using REV yielding
|     IF Null(cons(x,nil))
|        THEN cons(x,nil)
|        ELSE (rev(cdr(cons(x,nil)))*cons(car(cons(x,nil)),nil))
|
Trying to simplify
|     IF Null(cons(x,nil))
|        THEN cons(x,nil)
|        ELSE (rev(cdr(cons(x,nil)))*cons(car(cons(x,nil)),nil))
|
failed
→→→Trying to simplify the condition
|     Null(cons(x,nil))
|
succeeded using CONS yielding
|     FALSE
|
popping up
Trying to simplify
|     IF FALSE
|        THEN cons(x,nil)
|        ELSE (rev(cdr(cons(x,nil)))*cons(car(cons(x,nil)),nil))
|
succeeded using LOGICTREE yielding
|     (rev(cdr(cons(x,nil)))*cons(car(cons(x,nil)),nil))
|
Trying to simplify
|     (rev(cdr(cons(x,nil)))*cons(car(cons(x,nil)),nil))
|
1  |while trying to match *, SORT scruples do not permit me to bind u
|     to rev(cdr(cons(x,nil)))
|
→→→Trying to simplify argument 1
|        rev(cdr(cons(x,nil)))
|
|while trying to match rev, SORT scruples do not permit me to bind u
|        to cdr(cons(x,nil))
|
→→→Trying argument 1
|        cdr(cons(x,nil))
|
|        nil
|
popping up
Trying to simplfy
2       |        rev(nil)
|
succeeded using REV yielding
|        IF Null(nil) THEN nil ELSE (rev(cdr(nil))*cons(car(nil),nil))
|
popping up
Trying to simplify
|     (IF Null(nil) THEN nil ELSE (rev(cdr(nil))*cons(car(nil),nil))*
|        cons(car(cons(x,nil),nil)))
|
3  |while trying to match *, SORT scruples do not permit me to bind u
|     to IF Null(nil) THEN nil ELSE rev(cdr(nil)*cons(car(nil),nil)
|
→→→Trying to simplify argument 1
```

```
        |       IF Null(nil THEN nil ELSE rev(cdr(nil)*cons(car(nil),nil)
        |
        failed
        →→→Trying to simplify condition
              |  Null(nil)
              |
              succeeded using line 1 uielding
              | TRUE
              |
        popping up
        Trying to simplify
        |       IF TRUE THEN nil ELSE rev(cdr(nil)*cons(car(nil),nil)
        |
        succeeded using LOGICTREE yielding
        |       nil
  popping up
  Trying to simplify
  |       nil*cons(car(cons(x,nil)),nil)
  |
4 |while trying to match *, SORT scruples do not permit me to bind u
  |       to cons(car(cons(x,nil)),nil)
  |
  →→→Trying to simplify argument 1
        | nil
        |
5       failed but we are at a leaf: argument 1 completely simplified
  popping up
  →→→Trying to simplify argument 2
        |       cons(car(cons(x,nil)),nil)
        |
        failed
        →→→Trying to simplify argument 1
              | car(cons(x,nil))
              |
              succeeded using CAR yielding
              | x
              |
        popping up
        Trying to simplify
        |       cons(x,nil)
        |
        failed
        →→→Trying to simplify argument 1
              | x
              |
              failed but we are at a leaf: argument 1 completely simplified
        popping up
        →→→Trying to simplify argument 2
              | nil
              |
              failed but we are at a leaf: argument 2 completely simplified
        popping up
        argument 2 completely simplified
        popping up
        Trying to simplify
  |       nil*cons(x,nil)
  |
  succeeded using APPEND yielding
  |       IF Null(nil) THEN cons(x,nil) ELSE cons(car(nil),(cdr(nil)*cons(x,nil)))
  |
  Trying to simplify
  |       IF Null(nil) THEN cons(x,nil) ELSE cons(car(nil)*cons(x,nil)))
  |
  failed
  →→→Trying to simplify condition
        |       Null(nil)
        |
        succeeded using line 1 yielding
        |       TRUE
        |
  popping up
  Trying to simplify
  |       IF TRUE THEN cons(x,nil) ELSE cons(car(nil),(cdr(nil)*cons(x,nil)))
  |
  succeeded using LOGICTREE yielding
  |       cons(x,nil)
  |
  Trying to simplify
  |       cons(x,nil)
  |
6 this node already maximally simplified
  return cons(x,nil)
  11 substitutions were made
  26 calls were made to SIMPLIFY
```

Note 1. This is the FOL sort checking mechanism at work. FOL knows that x is an Sexp (by declaration) and that nil is a List because nil is of sort Null and Lists are more general than Nulls. This means that it knows by declaration that cons(x,nil) is a List. Unfortunately, it knows nothing about the cdr of a List. Thus since the definition of rev requires that u be instantiated to a Lists, this attempted replacement fails, and we try to simplify its arguments.

Note 2. Notice that the argument to rev actually simplifies to something that FOL can recognize as a List. This means that sort scruples do not prohibit the instantiation of the definition of rev.

Note 3. Unfortunately we have the same problem as in Note 1.

Note 4. This time the first argument to * is ok, but the second is not. Again we try to simplify the arguments.

Note 5. This time when we try to simplify nil nothing happens. In this case as a subterm it is completely simplified and gets marked in such a way that the simplifier never tries to do this again.

Note 6. It is very clever and remembers that it saw this before and since it is at the top level with a maximally simplified formula it stops.

F. An example of evaluation

This is an abbreviated trace of the evaluation of fact (2).

```
eval
|    fact (2)

interpreting
|    fact
fails

→→→Syntactic simplification succeeds, yielding
   | IF   2 = ∅ THEN 1 ELSE 2*fact(pred(2))
   |
   eval
   | IF   2 = ∅ THEN 1 ELSE 2*fact(pred(2))
   |
   →→→eval
      | 2 = ∅
      semantic evaluation succeeds, yielding
      | FALSE
   popping up
   semantic evaluation succeeds, yielding
   | 2*fact(pred(2))
   |
   interpreting
   | *
   succeeds evaluating args
     1   eval
         | 2
         semantic evaluation succeeds, yielding
         | 2
         |
     2   eval
         | fact(pred(2))
         |
         interpreting
         | fact
         fails
         |
         Syntactic simplification succeeds, yielding
         | IF pred(2) = ∅ THEN 1 ELSE pred(2)*fact(pred(pred(2)))
         |
         →→→eval
            | F pred(2) = ∅
            semantic evaluation succeeds, yielding
            | FALSE
   popping up
   semantic simplification succeeds, yielding
   | pred(2)*fact(pred(pred(2)))
   |
   eval
   | pred(2)*fact(pred(pred(2)))
   |
   interpreting
   | *
   succeeds evaluating args]
     1   eval
         | pred(2)
         semantic simplification succeeds, yielding
         | 1
         |
     2   eval
         | fact(pred(pred(2)))
         |
         interpreting
         | fact
         fails
         |
         Syntactic simplification succeeds, yielding
         | IF pred(pred(2)) = ∅
         |    THEN 1 ELSE pred(pred(2))*fact(pred(pred(pred(2))))
         |
         eval
         | IF pred(pred(2)) = ∅
         |  THEN 1 ELSE pred(pred(2))*fact(pred(pred(pred(2))))
         |
         eval
         | pred(pred(2)) = ∅
         semantic evaluating succeeds, yielding
         | TRUE
         |
         semantic evaluation succeeds, yielding
         | 1
         |
   Evaluating 1 gives 1
 Evaluating IF pred(pred(2)) = ∅ THEN 1 ELSE pred(pred(2))*fact(pred(pred
                                                          (pred(2)))) gives 1
Evaluating fact(pred(pred(2))) gives 1
Evaluating pred(2)*fact(pred(pred(2))) gives 1
Evaluating IF pred(2) = ∅ THEN 1 ELSE pred(2)*fact(pred(pred(2))) gives 1
Evaluating fact(pred(2)) gives 1
Evaluating 2*fact(pred(2)) gives 2
Evaluating IF 2 = ∅ THEN 1 ELSE 2*fact(pred(2)) gives 2
Evaluating fact(2) gives 2
1 fact(2) = 2
```

REFERENCES

1. Aristotle (-350) Organan.
2. Boyer, R. S. and Moore, J. S. (1979), *A Computational Logic*. To be published in the ACM Monograph Series (Academic Press).
3. Bulnes, J. (1978), GOAL: A goal oriented command language for interactive proof construction forthcoming Ph.D. thesis, Stanford University, Stanford.
4. Burstall, R. M. and Darlington, J. (1977), A transformation system for developing recursive programs, *JACM* **24** (1), 44–67.
5. Cartwright, R. (1977), Practical formal semantic definition and verification systems, Ph.D. thesis, Stanford University, Stanford.
6. Davis, M. and Schwartz, J. T. (1977), Correct-program technology/extensibility of verifiers–Two papers on program verification, Courant Computer Science Report #12, New York University.
7. Diffie, W. (1973), PCHECK: operation of the proof checker, unpublished.
8. Feferman, S. (1962), Transfinite recursive progressions of axiomatic theories, *J. Symbolic Logic* **27**, 259–316.
9. Filman, R. E. and Weyhrauch, R. W. (1976), A FOL Primer, Stanford Artificial Intelligence Laboratory Memo AIM-228, Stanford University, Stanford.
10. Filman, R. E. (1978), The interaction of observation and inference, forthcoming Ph.D. thesis, Stanford University, Stanford.
11. Green, C. (1969), The application of theorem proving to question-answering systems, Ph.D. thesis, Stanford University, Stanford.
12. Kelley, J. L. (1955), *General topology* (Van Nostrand, Princeton, NJ.) 298 pp.
13. Kleene, S. C. (1952), *Introduction to Metamathematics* (Van Nostrand, Princeton, NJ.) 550 pp.
14. Kleene, S. C. (1967), *Mathematical Logic* (Wiley, New York) 398 pp.
15. Kowalski, R. (1974), Predicate logic as a programming language, *Proc. IFIP Congress* 1974.
16. Kreisel, G. (1971a), Five notes on the application of proof theory to computer science, Stanford University: IMSSS Technical Report 182, Stanford.
17. Kreisel, G. (1971b), A survey of proof theory, II in: Fenstad, J. E. (Ed.), *Proc. the Second Scandinavian Logic Symposium* (North-Holland, Amsterdam).
18. McCarthy, J. (1963), A basis for a mathematical theory of computation, in: Computer Programming and Formal Systems (North-Holland, Amsterdam).
19. McCarthy, J. and Hayes, P. J. (1969), Some philosophical problems from the viewpoint of Artificial Intelligence, in: Michie, D. (Ed.), *Machine Intelligence* **7** (Edinburgh U.P., Edinburgh).
20. McCarthy, J. (1973), appendix to: Diffie. W., PCHECK: Operation of the proof checker. Unpublished.
21. McCarthy, J. (1978), Representation of recursive programs in first order logic, in: *Proc. International Conference on Mathematical Studies of Information Processing, Kyoto, Japan.*
22. Prawitz, D. (1965), *Natural Deduction—a proof-theoretical study* (Almqvist & Wiksell, Stockholm).
23. Robinson, R. M. (1950), An essentially undecidable axiom system, in: *Proc. Int. Cong. Math.* **1**, 729–730.
24. Royden, H. L. (1963), *Real Analysis* (Macmillan, New York).
25. Warren, D. (1977), *Implementing PROLOG—compiling predicate logic programs*, Vol. 1 and Vol. 2, DAI Research Reports Nos. 39 and 40, Edinburgh.
26. Weyhrauch, Richard W. (1977), FOL: A proof checker for first-order logic, Stanford Artificial Intelligence Laboratory Memo AIM-235.1, Stanford University, Stanford.
27. Weyhrauch, Richard W. (1978), Lecture notes on the use of logic in artificial intelligence and mathematical theory of computation, Summer school on the foundations of artificial intelligence and computer science (FAICS), Pisa.

The following series of notes refers to my working papers which are sometimes available from me.

[Note 6]. Weyhrauch, Richard W., FOL: a reasoning system, Informal Note 6. Unpublished.
[Note 15]. Weyhrauch, Richard W., The logic of FOL, Informal Note 15. Unpublished.

Subjective Bayesian methods for rule-based inference systems*

by RICHARD O. DUDA, PETER E. HART and NILS J. NILSSON
Stanford Research Institute
Menlo Park, California

ABSTRACT

The general problem of drawing inferences from uncertain or incomplete evidence has invited a variety of technical approaches, some mathematically rigorous and some largely informal and intuitive. Most current inference systems in artificial intelligence have emphasized intuitive methods, because the absence of adequate statistical samples forces a reliance on the subjective judgment of human experts. We describe in this paper a subjective Bayesian inference method that realizes some of the advantages of both formal and informal approaches. Of particular interest are the modifications needed to deal with the inconsistencies usually found in collections of subjective statements.

INTRODUCTION

One of the characteristics of human reasoning is the ability to form useful judgments from uncertain and incomplete evidence. This ability is not only needed for everyday activities, which people would normally never formalize, but also for tasks such as medical diagnosis or securities analysis, which have been subjected to formal treatment.

Because the general need to form judgments from incomplete data is so widespread, many techniques have been developed to aid or supplant people in this task. Probability theory and statistics provide a powerful framework for dealing with many inference problems.[1,2] In standard approaches, the link between alternative hypotheses and relevant evidence is represented by conditional or joint probabilities that are estimated from statistical samples. If the number of alternative hypotheses and the amount of relevant evidence are not too great, and if the available sample is sufficiently large, then probability and statistics furnish the preferred analytical tools. However, when many kinds of evidence simultaneously bear on an hypothesis, traditional statistical approaches become inappropriate because estimation problems become unmanageable.

Recent work in artificial intelligence has suggested other approaches to the problem of resolving hypotheses on the basis of a mass of uncertain evidence. Among the most attractive are *rule-based systems*, which use a large body of *inference rules*, supplied by experts, to provide the knowledge needed to distinguish among competing hypotheses.[3-6] Each inference rule defines the role of a particular set of evidence in resolving a particular hypothesis. Typically, an ad hoc scoring function is used to combine the effects of collections of uncertain evidence acting through several inference rules on the same hypothesis. Thus, rule-based systems attempt to substitute judgments distilled from long experience for joint probabilities estimated from prohibitively large samples.

Our purpose in this paper is to describe a subjective Bayesian technique that can be used in place of ad hoc scoring functions in rule-based inference systems. Our intent is to retain insofar as possible the well-understood methods of probability theory, introducing only those modifications needed because we are dealing with networks of subjective inference rules. The scope of the paper is limited; we shall not discuss here the more general issues of representation and control that must be faced when designing a complete rule-based inference system.

FUNDAMENTALS

In a rule-based inference system, the rules are typically of the form

$$\textit{If } E_1 \textit{ and } E_2 \textit{ and } \ldots \textit{ and } E_n \textit{ then } H$$

where $E_i (i=1 \ldots n)$ is the i^{th} piece of evidence and H is an hypothesis suggested by the evidence. Each inference rule has a certain *strength* measured by parameters that will be defined later. For now it suffices to say that the greater the strength, the greater is the power of the evidence to confirm the hypothesis. In most applications, the rules and their strengths are provided by carefully interviewing experts.

The individual pieces of evidence (the E_i) and the hypothesis (H) of a rule are propositional statements. Instead of being either absolutely true or false, the truth values of these propositional statements may be

* The work reported herein was supported by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-75-C-0005.

uncertain. In this paper we shall represent these uncertainties by probabilities, so that associated with each propositional statement is a corresponding probability value.

To simplify matters, we shall assume (without loss of generality) that each rule has only a single propositional statement as evidence on its left-hand side. To reduce a conjunction to a single statement, we need a method for computing the joint probability, $P(E_1, \ldots, E_n)$ from the individual probabilities $P(E_i)$. Two simple alternatives are to assume independence of the E_i or to use the fuzzy set computation $P(E_1, \ldots, E_n) = \min P(E_i)$. More generally, the left-hand side of a rule could contain an arbitrary logical expression, E. The results of this paper do not depend on how the probability of E is computed.

We represent a rule of the form "*if* E *then* H" graphically by the following structure:

Here a propositional statement is being represented as a node, and an inference rule is being represented as an arc. A collection of rules about some specific subject area invariably uses the same pieces of evidence to imply several different hypotheses. It also frequently happens that several alternative pieces of evidence imply the same hypothesis. Furthermore, there are often chains of evidences and hypotheses. For these reasons it is natural to represent a collection of rules as a graph structure or *inference net.*

An example of an inference net is shown in Figure 1.

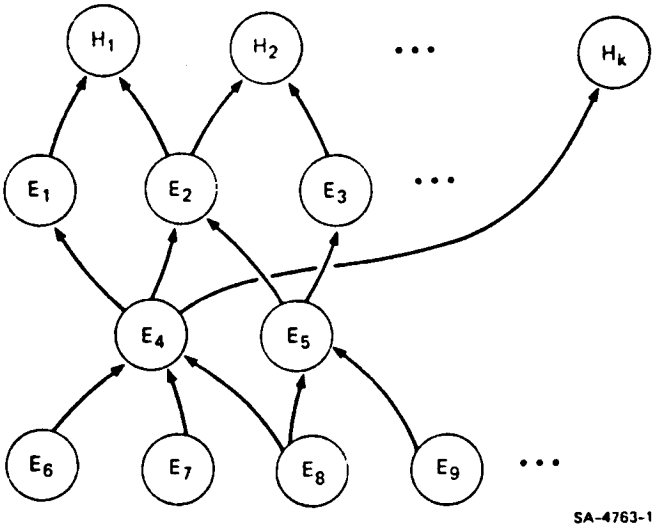

Figure 1—A simple inference net

The H_i at the top of the net are alternative hypotheses to be resolved. Each arc entering a node represents an inference rule and has associated with it a strength. Notice that a typical intermediate node like E_5 can play two roles: it provides supporting evidence for the nodes above it (E_2 and E_3), and it acts as an hypothesis to be resolved by evidence below it (E_8 and E_9).

The main problem to be considered in this paper concerns the propagation of probabilities through the net. Suppose for example, that a user of the net provides evidence by deciding that the probability of a node, say E_6, should be changed from its prior value to some new value. Obviously this should require updating of the probability of E_4 and, in turn, E_1, E_2, and H_k, and so on. Any mechanism used for propagating probabilities must be able to cope with a number of problems. The rules have uncertainty associated with them, and the evidence provided by a user may be uncertain. These two different kinds of uncertainty must somehow be combined. Multiple evidence typically bears on a single hypothesis, so that some form of independence must usually be assumed. Finally, the rules are provided subjectively by experts, so certain kinds of inconsistencies arise that can seriously jeopardize success. In the following sections we suggest a Bayesian updating scheme that addresses these concerns.

SUBJECTIVE BAYESIAN UPDATING

Suppose we are given a rule *if* E, *then* H. Let us begin with the simplified problem of updating the probability of H given its prior value and given that E is observed to be true. By Bayes rule, we have

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}. \tag{1}$$

For our purposes, a more convenient form of Bayes rule is arrived at by writing the complementary form for the negation of H

$$P(\bar{H}|E) = \frac{P(E|\bar{H})P(\bar{H})}{P(E)}, \tag{2}$$

and dividing Eq. (1) by Eq. (2), to obtain

$$\frac{P(H|E)}{P(\bar{H}|E)} = \frac{P(E|H)}{P(E|\bar{H})}\frac{P(H)}{P(\bar{H})}. \tag{3}$$

Each of the three terms in this equation has a traditional interpretation. We define the *prior odds* on H to be

$$O(H) = \frac{P(H)}{P(\bar{H})} = \frac{P(H)}{1-P(H)} \tag{4}$$

and the *posterior odds* to be

$$O(H|E) = \frac{P(H|E)}{P(\bar{H}|E)} = \frac{P(H|E)}{1-P(H|E)}. \tag{5}$$

Now the *likelihood ratio* is defined by

$$\lambda = \frac{P(E|H)}{P(E|\bar{H})}, \tag{6}$$

so Eq. 3 becomes the *odds-likelihood* formulation of Bayes rule:

$$O(H|E) = \lambda O(H). \tag{7}$$

This equation tells us how to update the odds on H given the observation of E. For rule-based inference systems, we assume that a human expert has given the rule and has provided the likelihood ratio λ to indicate the "strength" of the rule. A high value of $\lambda(\lambda \gg 1)$ represents, roughly speaking, the fact that E is sufficient for H, since the observation that E is true will transform indifferent prior odds on H into heavy posterior odds in favor of H. Notice, incidentally, that the underlying probabilities can be recovered from their odds by the simple formula

$$P = \frac{O}{O+1}, \tag{8}$$

so that the odds and the probabilities give exactly the same information.

Suppose now that we wish to update the odds on H given that E is observed to be false. In a strictly analogous fashion, we write.

$$O(H|\bar{E}) = \bar{\lambda} O(H), \tag{9}$$

where we define λ by

$$\lambda = \frac{P(\bar{E}|H)}{P(\bar{E}|\bar{H})} = \frac{1-P(E|H)}{1-P(E|\bar{H})}. \tag{10}$$

Notice that $\bar{\lambda}$ must also be provided by the human expert; it cannot be derived from λ. A low value of $\bar{\lambda}$, $(0 \leq \bar{\lambda} \ll 1)$ represents, roughly speaking, the fact that E is necessary for H, since the observation that E is false will by Eq. 9 transform indifferent prior odds on H into odds heavily against H. Curiously, although λ and $\bar{\lambda}$ must be separately provided by the expert, they are not completely independent of each other. In particular, Eqs. (6) and (10) yield

$$\bar{\lambda} = \frac{1-\lambda P(E|\bar{H})}{1-P(E|\bar{H})}, \tag{11}$$

so that, if we exclude the extreme cases of $P(E|\bar{H})$ being either 0 or 1, we see that $\lambda > 1$ implies $\bar{\lambda} < 1$, and $\lambda < 1$ implies $\bar{\lambda} > 1$. Further, we have $\lambda = 1$ if and only if $\bar{\lambda} = 1$. This means that if the expert gives a rule such that the presence of E enhances the odds on H (i.e., $\lambda > 1$), he should also tell us that the absence of E depresses the odds on H (i.e., $\bar{\lambda} < 1$). To some extent, this mathematical requirement does violence to intuition. People who work with rule-based inference systems are commonly told by experts that "The presence of E enhances the odds on H, but the absence of E has no significance." In other words, the expert says that $\lambda > 1$, but $\bar{\lambda} = 1$. Subsequently, we shall suggest some modifications that address this and other problems of inconsistency.

We note in passing that knowledge of both λ and $\bar{\lambda}$ is equivalent to knowledge of both $P(E|H)$ and $P(E|\bar{H})$. Indeed, it follows at once from Eqs. (6) and (10) that

$$P(E|H) = \lambda \frac{1-\bar{\lambda}}{\lambda - \bar{\lambda}} \tag{12}$$

and

$$P(E|\bar{H}) = \frac{1-\bar{\lambda}}{\lambda - \bar{\lambda}} \tag{13}$$

Thus, whether the expert should be asked to provide λ and $\bar{\lambda}$, $P(E|H)$ and $P(E|\bar{H})$, or, indeed, some other equivalent information is a psychological rather than a mathematical question.[7]

UNCERTAIN EVIDENCE AND THE PROBLEM OF PRIOR PROBABILITIES

Having seen how to update the probability of an hypothesis when the evidence is known to be either certainly true or certainly false, let us consider now how updating should proceed when the user of the system is uncertain. We begin by assuming that when a user says "I am 70 percent certain that E is true," he means that $P(E|$ relevant observations$) = .7$. We designate by E′ the relevant observations that he makes, and simply write $P(E|E')$ for the user's response.

We now need to obtain an expression for $P(H|E')$. Formally,

$$\begin{aligned} P(H|E') &= P(H,E|E') + P(H,\bar{E}|E') \\ &= P(H|E,E')P(E|E') \\ &\quad + P(H|\bar{E},E')P(\bar{E}|E'). \end{aligned} \tag{14}$$

We make the reasonable assumption that if we *know* E to be true (or false), then the observations E′ relevant to E provide no further information about H. With this assumption, Eq. (14) becomes

$$P(H|E') = P(H|E)P(E|E') + P(H|\bar{E})P(\bar{E}|E'). \tag{15}$$

Here $P(H|E)$ and $P(H|\bar{E})$ are obtained directly from Bayes rule, i.e., from Eq. (7) and Eq. (9), respectively.

If the user is certain that E is true, then $P(H|E') = P(H|E)$. If the user is certain that E is false, then $P(H|E') = P(H|\bar{E})$. In general, Eq. (15) gives $P(H|E')$ as a linear interpolation between these two extreme cases. In particular, note that if $P(E|E') = P(E)$ then $P(H|E') = P(H)$. This has the simple interpretation that if the evidence E′ is no better than a priori knowledge, then application of the rule leaves the probability of H unchanged.

In a pure Bayesian formulation, Eq. (15) is the solution to the updating question. In practice, however, there are significant difficulties in using this formulation in an inference net. These difficulties stem from a combination of the classical Bayesian dilemma over prior probabilities and the use of subjective probabilities.

To appreciate the difficulty, consider again a typical pair of nodes E and H embedded in an inference net. It is apparent from Eqs. (7) and (9) that the updating procedure depends on the availability of the prior odds

O(H). Thus, although we have not emphasized the point until now, we see that the expert must be depended upon to provide the prior odds as well as λ and $\bar{\lambda}$ when the inference rule is given. On the other hand, recall our earlier observation that E also acts as an hypothesis to be resolved by the nodes below it in the net. Thus, the expert must also provide prior odds on E. If all of these quantities were specified consistently, then the situation would be as represented in Figure 2. The straight line plotted is simply Eq. (15), and shows the interpolation noted above. In particular, note that if the user asserts that $P(E|E') = P(E)$, then the updated probability is $P(H|E') = P(H)$. In other words, if the user provides no new evidence, then the probability of H remains unchanged.

In the practical case, unfortunately, the subjectively obtained prior probabilities are virtually certain to be inconsistent, and the situation becomes as shown in Figure 3. Note that P(E), the prior probability provided by the expert, is different from $P_c(E)$, the probability consistent with P(H). Here, if the user provides no new evidence—i.e., if $P(E|E') = P(E)$—then the formal Bayesian updating scheme will substantially change the probability of H from its prior value P(H). Furthermore, for the case shown in Figure 3, if the user asserts that E is true with a probability $P(E|E')$ lying in the interval between P(E) and $P_c(E)$, then the updated probability $P(H|E')$ will be less than P(H). Thus, we have here an example of a rule intended to increase the probability of H if E is found to be true, but which turns out to have the opposite effect. This type of error can be compounded as probabilities are propagated through the net.

Several measures can be taken to correct the unfortunate effects of priors that are inconsistent with inference rules. Since the problem can be thought of as one of overspecification, one approach would be to relax

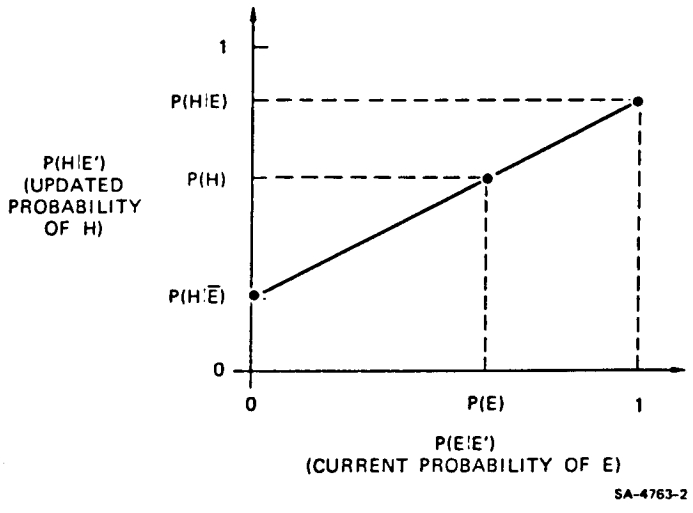

Figure 2—Idealized updating of P(H|E')

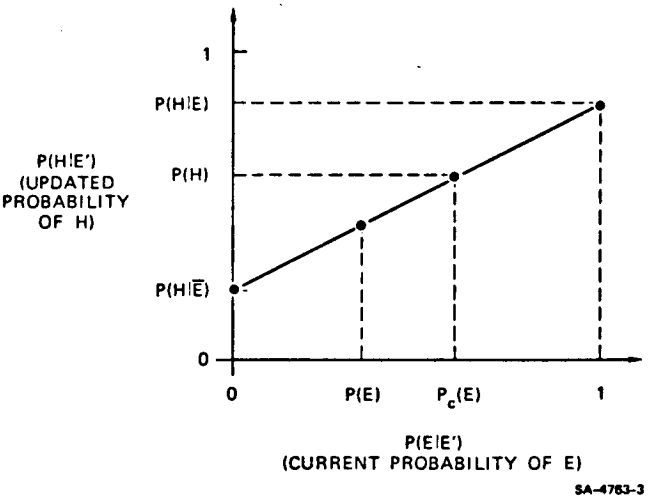

Figure 3—Inconsistent priors

the specification of whatever quantities are subjectively least certain. For example, if the subjective specification of P(E) were least certain (in the expert's opinion), then we might set $P(E) = P_c(E)$. This approach leads to difficulties because the pair of nodes E and H under consideration are embedded in a large net. For example, in Figure 1, we might be considering node E_2 as the hypothesis H, and node E_5 as the evidence E. If we were to establish a prior probability $P(E_5)$ to be consistent with $P(E_2)$, we would simultaneously make $P(E_5)$ inconsistent with the priors on E_8 and E_9, which provide supporting evidence for E_5. Prior probabilities can therefore not be forced into consistency on the basis of the local structure of the inference net; apparently, a more global process—perhaps a relaxation process—would be required.

A second alternative for achieving consistency would be to adjust the linear interpolation function shown in Figure 3. There are several possibilities, one of which is illustrated in Figure 4a. The linear function has been broken into a piecewise linear function at the coordinates of the prior probabilities, forcing consistent updating of the probability of H given E'. Two other possibilities are shown in Figures 4b and 4c. In Figure 4b we have introduced a dead zone over the interval between the specified prior probability P(E) and the consistent prior $P_c(E)$. Intuitively, the argument in support of this consistent interpolation function is that if the user cannot give a response outside this interval, then he is not sufficiently certain of his response to warrant any change in the probability of H. Figure 4c shows another possibility, motivated by the earlier observation that experts often give rules of the form "The presence of E enhances the odds on H, but the absence of E has no significance." By keeping $P(H|E')$ equal to P(H) when $P(E|E')$ is less than P(E) we are

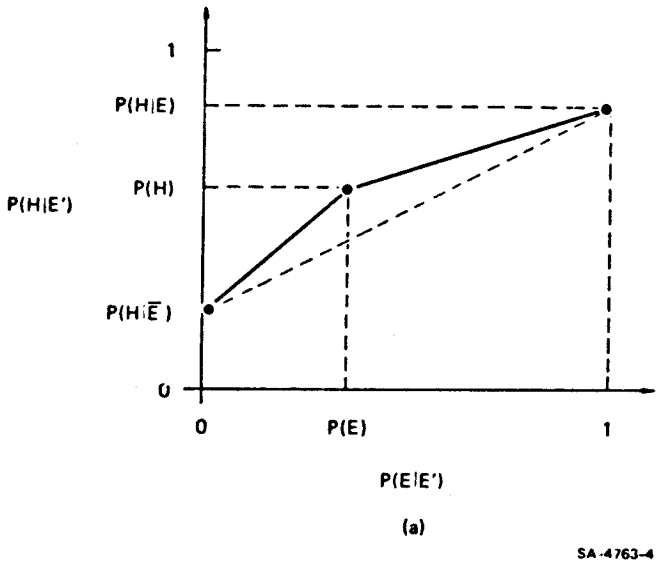

Figure 4(a)—Consistent interpolation functions

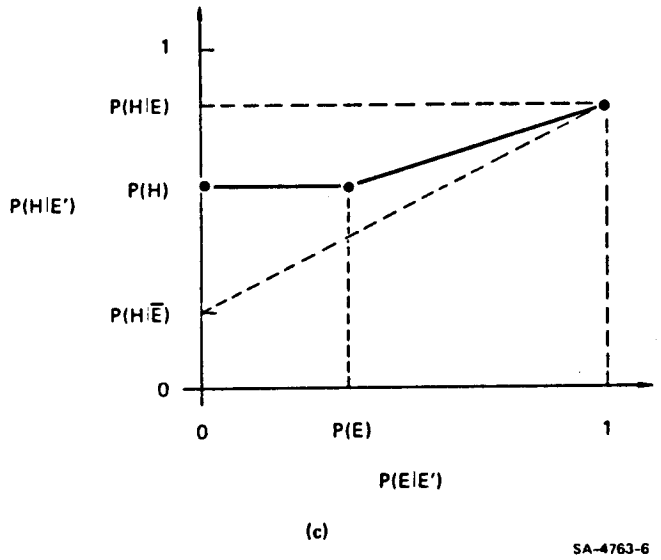

Figure 4(c)—Consistent interpolation functions (concluded)

effectively allowing the forbidden situation where $\lambda > 1$ and $\bar{\lambda} = 1$. In effect, this is equivalent to the method illustrated in Figure 4a under the assumption that $P(H|\bar{E}) = P(H)$.

It is interesting to compare these modifications with the procedure used by Shortliffe to handle uncertain evidence in the MYCIN system.[4,5] While the nonlinear equations that result from use of Shortliffe's version of confirmation theory prevent a general comparison, it is possible to express his procedure in our terms for the special case of a single rule. The result for the case in which the presence of E supports H is shown in Figure 5. Clearly, the solution is identical to that of Figure 4c except for the interval from P(E) to $P_t(E)$ within which Shortliffe's solution maintains P(H|E') at the a priori value P(H).

The graphical representations in Figures 2 through 4 provide a nice vehicle for visualizing the discrepancies between formal and subjective Bayesian updating, and make it easy to invent other alternatives for reconciling inconsistencies. For completeness, the Appendix contains the easily computable algebraic representa-

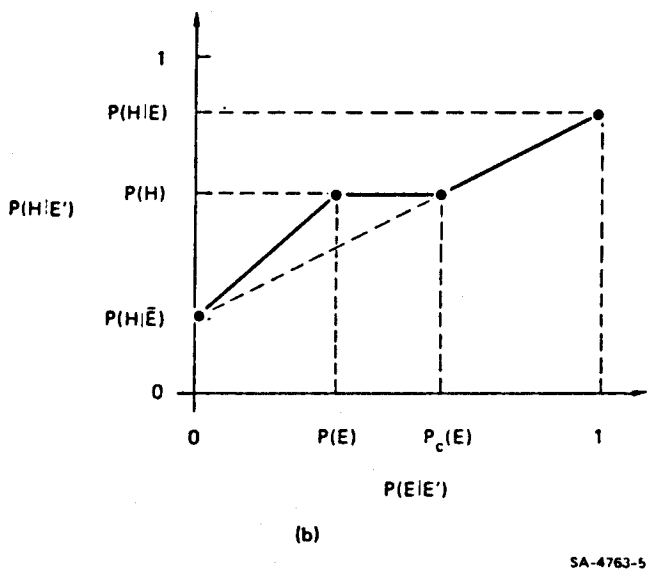

Figure 4(b)—Consistent interpolation functions (continued)

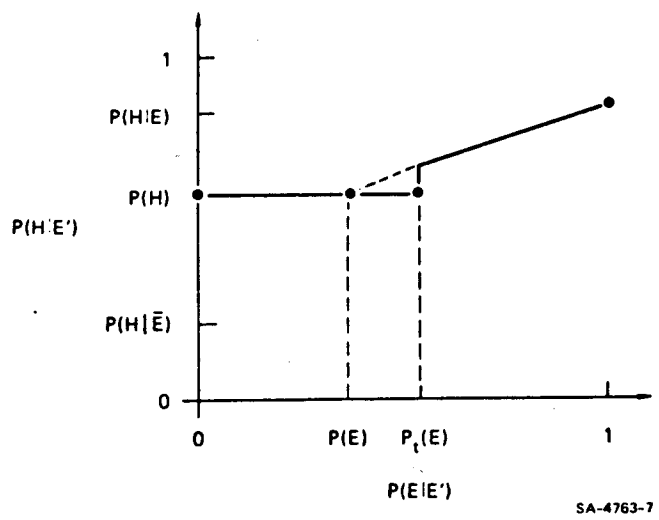

Figure 5—The interpolation function used in the mycin system $P_t(E) = P(E) + t[1 - P(E)]$. Typically, $t = 0.2$.

tions of these functions, and also treats the complementary case in which the straight line given by Eq. (15) has a negative slope (the case in which $\lambda<\bar{\lambda}$). In a small experimental system, the function shown in Figure 4a has given satisfactory preliminary results.[8]

THE USE OF MULTIPLE EVIDENCE

We turn now to the more general updating problem in which several rules of the form $E_1 \rightarrow H, \ldots, E_n \rightarrow H$ all concern the same hypothesis H.* Since most nodes in actual inference nets have several incoming arcs, this is the case of greatest practical interest. In order to gain some insight about how multiple evidence should be used to update H when the evidence is uncertain and the priors are inconsistent, let us first consider briefly how updating would formally proceed in simpler cases.

Suppose the i^{th} inference rule has associated with it the usual two quantities λ_i and $\bar{\lambda}_i$. For a first simple case, how should H be updated when all the E_i have been observed to be certainly true? This case is analogous to the case summarized by Eq. (7). Under the assumption that the pieces of evidence are conditionally independent (i.e., that $P(E_1, \ldots, E_n|H) = \prod_{i=1}^{n} P(E_i|H)$ and that $P(E_1, \ldots, E_n|\bar{H}) = \prod_{i=1}^{n} P(E_i|\bar{H})$), it is not difficult to reach an analogous answer. Specifically, the odds on H are updated by the expression

$$O(H|E_1, \ldots, E_n) = \left[\prod_{i=1}^{n} \lambda_i\right] O(H), \qquad (16)$$

where

$$\lambda_i = \frac{P(E_i|H)}{P(E_i|\bar{H})}. \qquad (17)$$

Similarly, if all the evidence is observed to be certainly false, we can under conditional independence assumptions again factor the joint likelihood ratio to obtain

$$O(H|\bar{E}_1, \ldots, \bar{E}_n) = \left[\prod_{i=1}^{n} \bar{\lambda}_i\right] O(H). \qquad (18)$$

Now let us consider the general case of uncertain evidence and inconsistent prior probabilities. We already know that the posterior odds $O(H|E_i')$ given a single observation E_i' can be computed using updating functions like the ones shown in Figure 4. We can therefore define, for a single inference rule, an effective likelihood ratio λ_i' by

$$\lambda_i' \triangleq \frac{O(H|E_i')}{O(H)}. \qquad (19)$$

* This should not be confused with the conjunctive premise mentioned earlier.

By making the assumption now that the E_i' are independent, we can obtain for the general case an expression similar to the simple updating formulas given by Eqs. (16) and (18):

$$O(H|E_1', \ldots, E_n') = \left[\prod_{i=1}^{n} \lambda_i'\right] O(H). \qquad (20)$$

To use this expression in an inference net system, we simply store with each node its prior odds (or probability), and store with each incoming arc an effective likelihood ratio λ_i'. Whenever a piece of evidence provided by the user causes $P(E_i|E_i')$ to be updated, a new effective likelihood ratio is computed and the posterior odds in favor of H is computed using Eq. (20). This procedure has the following consequences:

(1) If no evidence is obtained for a rule, then it will retain an initial effective likelihood ratio of unity, since prior and "posterior" odds are the same.
(2) The order in which evidence is obtained and rules are applied does not affect the final posterior probabilities.
(3) The same rule can be used repeatedly, with the same or different values for the probability of the evidence. In particular, if a user changes his mind and modifies an earlier assertion, the new assertion will correctly "undo" any effects of earlier statements.

CONCLUSIONS

The probability updating procedure presented here has several points to recommend it. It accepts subjective information that can readily be obtained from experts. The two conditional probabilities, $P(E|H)$ and $P(E|\bar{H})$, that determine the strength of an inference rule typically are intuitively meaningful measures, and the procedure is tolerant of the inevitable inconsistencies in subjective expert information. The basis in probability theory of our procedure provides a useful theoretical foundation for calculating the effects of uncertain evidence. One value of theory is that it makes us explicitly aware of certain underlying assumptions about such matters as conditional independence, prior probabilities, and inconsistent information. Finally, our procedure is straightforward computationally and can be readily implemented in inference net systems.

There are, however, some questions that remain to be dealt with. If the network contains multiple paths linking a given piece of evidence to the same hypothesis, the independence assumption is obviously violated. It is important to settle on a reasonable (if ad hoc) modification of our basic procedure that behaves appropriately in such situations. (A more extreme complication would involve being able to avoid the circular reasoning implied by inference nets with loops.)

There are sometimes cases where some of the nodes in an inference net are related by a constraint not expressed in any given rule. For example, a subset of hypotheses may be mutually exclusive and exhaustive, in which case their probabilities must always sum to one, regardless of their individual values. Such a constraint may be inconsistent with the associated rule strengths given us by the experts. Perhaps a simple expedient, such as renormalization of probability values, can be justified in this case.

We have not addressed here at all issues of inference net control strategy: for example, which hypotheses should be pursued and which evidence should be sought at any step. The answers to these sorts of questions may be heavily dependent on the particular application. Another global question concerns rules containing logical statements that may include quantifiers and variables. But in whatever way these questions are answered, the basic updating procedure presented here would appear to be a useful component of rule-based inference systems.

ACKNOWLEDGMENTS

We have benefited from the comments of many of our colleagues, but would like particularly to acknowledge the contributions of Georgia Sutherland at SRI, and the stimulating and helpful discussions with E. H. Shortliffe, Bruce Buchanan, Randall Davis, and Dana Ludwig at Stanford University. This work was supported by the Advanced Projects Research Agency under Contract DAHCO4-75-C-0005.

REFERENCES

1. Hadley, G., *Introduction to Probability and Statistical Decision Theory*, Holden-Day, San Francisco, California, 1967.
2. Raiffa, H., *Decision Analysis*, Addison-Wesley, New York, New York, 1968.
3. Waterman, D. A., "Generalization Learning Techniques for Automating the Learning of Heuristics," *Artificial Intelligence*, Vol. 1, pp. 121-170, Spring 1970.
4. Shortliffe, E. H., *MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection*, Stanford Artificial Intelligence Laboratory Memo AIM-251, Stanford University, Stanford, California, October 1974.
5. Shortliffe, E. H. and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences*, Vol. 23, pp. 351-379, 1975.
6. Davis, R. and J. King, "An Overview of Production Systems," in *Machine Representations of Knowledge*, D. Reidel Publishing Co.; forthcoming.
7. Gustafson, D. H., et al., "Wisconsin Computer Aided Medical Diagnosis Project—Progress Report," in *Computer Diagnosis and Diagnostic Methods*, pp. 255-278, J. A. Jacquez, ed., Charles C. Thomas, Springfield, Illinois, 1972.
8. Sutherland, G., *Implementation of Inference Nets—II*, Technical Note 122, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, January 1976.

APPENDIX

Complete analytical expressions giving $P(H|E')$ as a piecewise linear function of $P(E|E')$ are given in this Appendix. These expressions correspond to the three graphical representations illustrated in Figure 4. The simplest expression corresponds to Figure 4a:

$$P(H|E') = \begin{cases} P(H|\bar{E}) + \dfrac{P(E|E')}{P(E)}[P(H) - P(H|\bar{E})] & 0 \le P(E|E') \le P(E) \\ \dfrac{P(H) - P(H|E)P(E)}{1-P(E)} + P(E|E')\dfrac{P(H|E) - P(H)}{1-P(E)} & P(E) \le P(E|E') \le 1 \end{cases} \quad \text{(A1)}$$

Here it is important to note that the four quantities $P(H)$, $P(E)$, $P(H|E)$, and $P(H|\bar{E})$ are assumed to be estimates obtained from experts. Were the true probabilities to be used in this formula, it would reduce at once to the linear expression given by Eq. (15). The estimates of $P(H|E)$ and $P(H|\bar{E})$ might be obtained directly from an expert, but would more often be obtained through Bayes rule [Eqs. (7) and (9), respectively]. To be explicit,

$$P(H|E) = \frac{P(E|H)P(H)}{[P(E|H) - P(E|\bar{H})]P(H) + P(E|\bar{H})} = \frac{\lambda P(H)}{(\lambda - 1)P(H) + 1} \quad \text{(A2)}$$

and

$$P(H|\bar{E}) = \frac{[1 - P(E|H)]P(H)}{[P(E|\bar{H}) - P(E|H)]P(H) + 1 - P(E|\bar{H})} = \frac{\bar{\lambda} P(H)}{(\lambda - 1)P(H) + 1} \quad \text{(A3)}$$

To obtain the equations for Figure 4b, we define $P_c(E)$ by

$$P_c(E) = \frac{P(H) - P(H|\bar{E})}{P(H|E) - P(H|\bar{E})} \quad \text{(A4)}$$

In general, this quantity will differ from the $P(E)$ value supplied by the expert. For Figure 4b we must distinguish between the two cases $P(E) \le P_c(E)$ and $P(E) > P_c(E)$. The equations are as follows:

Case 1: $P(E) \leq P_c(E)$

$$P(H|E) = \begin{cases} P(H|\bar{E}) + \dfrac{P(E|E')}{P(E)}[P(H) - P(H|\bar{E})] & 0 \leq P(E|E') \leq P(E) \\ P(H) & P(E) \leq P(E|E') \leq P_c(E) \\ P(H|\bar{E}) + P(E|E')[P(H|E) - P(H|\bar{E})] & P_c(E) \leq P(E|E') \leq 1 \end{cases} \tag{A5}$$

Case 2: $P(E) > P_c(E)$

$$P(H|E') = \begin{cases} P(H|\bar{E}) + P(E|E')[P(H|E) - P(H|\bar{E})] & 0 \leq P(E|E') \leq P_c(E) \\ P(H) & P_c(E) \leq P(E|E') \leq P(E) \\ \dfrac{P(H) - P(H|E)P(E)}{1 - P(E)} + P(E|E')\dfrac{P(H|E) - P(H)}{1 - P(E)} & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A6}$$

Finally, there are also two cases to be distinguished for Figure 4c. The first case corresponds to assuming that $P(H|\bar{E}) \approx P(H)$, so that $P_c(E) \approx 0$. The second case corresponds to assuming that $P(H|E) \approx P(H)$, so that $P_c(E) \approx 1$. In effect, these cases correspond to the rules $E \xrightarrow{\lambda} H$ and $\bar{E} \xrightarrow{\bar{\lambda}} H$ taken separately. The corresponding equations are special cases of Eqs. (A5) and (A6):

Case 1: $E \xrightarrow{\lambda} H$

$$P(H|E') = \begin{cases} P(H) & 0 \leq P(E|E') \leq P(E) \\ \dfrac{P(H) - P(H|E)P(E)}{1 - P(E)} + P(E|E')\dfrac{P(H|E) - P(H)}{1 - P(E)} & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A7}$$

Case 2: $\bar{E} \xrightarrow{\bar{\lambda}} H$

$$P(H|E') = \begin{cases} P(H|\bar{E}) + \dfrac{P(E|E')}{P(E)}[P(H) - P(H|\bar{E})] & 0 \leq P(E|E') \leq P(E) \\ P(H) & P(E) \leq P(E|E') \leq 1 \end{cases} \tag{A8}$$

Ordinarily one would view this as a simplified approximation that is useful when one of the two likelihood ratios is dominant. However, it is interesting to observe that if both λ and $\bar{\lambda}$ are significant and if the two separate rules $E \xrightarrow{\lambda} H$ and $\bar{E} \xrightarrow{\bar{\lambda}} H$ are treated as if E and $\bar{E}$ were statistically independent, then Eqs. (A7) and (A8) yield the same result as Eq. (A1). This follows from the fact that when $P(H|E') = P(H)$ we have $O(H|E') = O(H)$, so that Eq. (19) yields $\lambda' = 1$. Thus, if $0 \leq P(E|E') \leq P(E)$ only the rule $\bar{E} \xrightarrow{\bar{\lambda}} H$ contributes to $P(H|E')$, while if $P(E) \leq P(E|E') \leq 1$ only the rule $E \xrightarrow{\lambda} H$ contributes to $P(H|E')$, the contributions being exactly those given in Eq. (A1).