Theorist: a logical reasoning system for defaults and diagnosis

David Poole
Randy Goebel
Romas Aleliunas

Research Report CS-86-06
February 1986

Theorist: a logical reasoning system for defaults and diagnosis†

David Poole
Randy Goebel
Romas Aleliunas

Logic Programming and Artificial Intelligence Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Abstract

We provide an introduction to Theorist, a logic programming system that uses a uniform deductive reasoning mechanism to construct explanations of *observations* in terms of *facts* and *hypotheses*. *Observations*, *facts*, and *possible hypotheses* are each sets of logical formulas that represent, respectively, a set of observations on a partial domain, a set of facts for which the domain is a model, and a set of tentative hypotheses which may be required to provide a consistent explanation of the *observations*.

Theorist has been designed to reason in a fashion similar to how we reason with and construct scientific theories. Rather than monotonically deduce theorems from a fixed logical theory, theorist distinguishes *facts* from *hypotheses* and attempts to use deduction to construct consistent theories for which the *observations* are logical consequences. A prototype, implemented in Prolog, demonstrates how diagnosis, default reasoning, and a kind of learning can all be based on the Theorist framework.

Introduction

Prolog is the *generic* resolution-based logic programming language. Its formulas consists of *facts*, e.g., *father(randy,kari)*, *rules*, e.g., *parent(P,C) if father(P,C)*, and *queries*, e.g., *parent(P,kari)?* Facts and rules comprise Prolog *assertions*, and are used to answer queries. Here the example fact and rule can be used to provide the answer that one parent of *kari* is *P=randy*.

As explained in the paper by Dahl (this volume), this class of logical formulas is called Horn clauses, and is appropriate as a representation language when the domain description requires no incomplete or negative knowledge. From this view, Prolog is an implementation of a proof procedure for a simple logic, where knowledge about the world is specified in terms of Horn Clauses, where variables are universally quantified, and answers are logical consequences of the knowledge.

†- this paper was submitted in September 1985 as a chapter of the book *Knowledge Representation*, N.J. Cercone and G. McCalla (eds.), Springer-Verlag, [in preparation].

There are, of course, other ways to view Prolog. It can be seen as

- a programming language, with pattern matching procedure calls and a backtracking control structure;
- a database language used to specify relations which are either stored or derived;
- as a question-answering system, where queries and bodies of clauses are treated as questions to be answered by heads of clauses. A clause is viewed as an assertion which means "to answer a question matching the head, answer the questions provided by the body.

Prolog is rather successful at these, and much research is being done to make it efficient for each of these uses. In fact, in any extension one might consider, it is desirable to retain these views and the efficiency with which Prolog provides them.

Prolog as a representation system

A serious deficiency of Prolog as a representation system is that answers are often not just logical consequences of our knowledge. For example,

- non-monotonic reasoning uses generalised knowledge, as long as it is consistent. New knowledge can then "remove" things previously deduced;
- diagnosis is not just finding what follows from our knowledge. A diagnosis is not a logical consequence of our observations about a patient. In fact exactly the opposite is the case: it is the observations that should be shown to be logical consequences of our knowledge and the diagnosis;
- learning by forming models of the world is, again, not just finding out what follows from what we know. The scientific method suggests that we build up theories about the world which are to be empirically verified;
- even many mundane tasks like filling out an income tax form require us to make uniform choices about how we declare some income. Your final tax payable is not just a logical consequence of your income, but how you choose to declare various kinds of income and deductions.

This paper suggests a way to solve all of these problems within a uniform framework. The proposed method is an extension to pure Prolog, and can exploit the implementation technology of current Prolog systems.

The Theorist framework

Here we provide an overview of the Theorist framework, in order that the following explanation of appropriate task domains is clear. Theorist is a system for representation and reasoning. The current representation language is the clausal form of first order logic. A typical theorist knowledge base consists of a collection of formulas classified as *possible hypotheses* (Δ), *facts* (F), and *observations* (G) (see figure 1). The Theorist reasoning strategy attempts to accumulate consistent sets of facts and instances of hypotheses as explanations for which the observations are logical consequences.

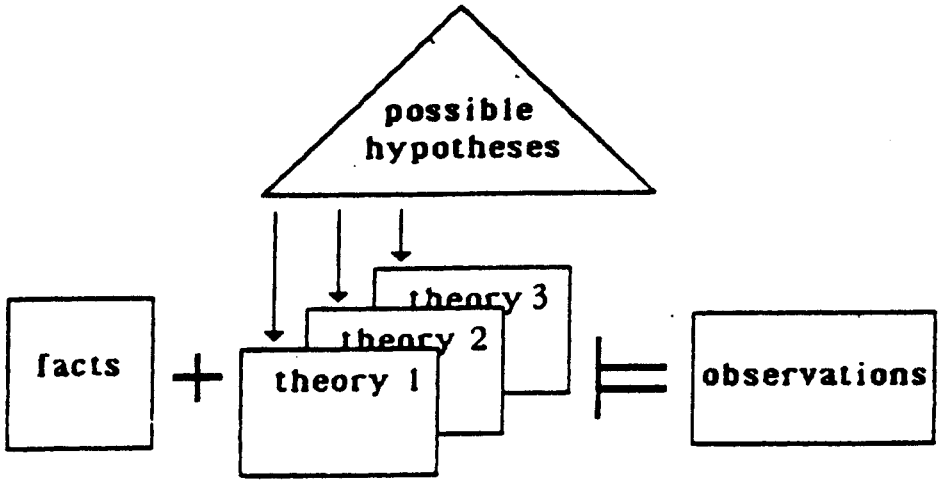

Figure 1 The Theorist framework

The three formulae we use are

F - the set of formulae we know are true in the world we are trying to represent. We assume that these are consistent (the intended interpretation being a model).

Δ - the set of possible hypotheses. These are things we are prepared to accept as part of an explanation. By allowing these to be defaults we get default reasoning, to let these be possible diseases or malfunctions we get diagnosis, and by letting these be generalisations of the observations or laws derived from the observations then we get learning.

G - the set of observations to be explained.

A theory is a subset of the possible hypotheses which are consistent, and imply the observations. More formally, we say G is *explainable* if there is some subset D of Δ such that

$F \cup D \models G$ and
$F \cup D$ is consistent

D is said to be a *theory that explains* G. D should be seen as a "scientific theory" [Popper58, Quine78].

The following sections show how by considering Δ in different ways we can form a unifying framework for many AI reasoning problems.

The following syntax will be used for facts and defaults (all examples in this paper work with our current Theorist prototype).

:*fact* <*clause*>;
:*default* <*name*> <*clause*>;

where <*clause*> is a clause as in Prolog, but with the symbol $n(a)$ being the true negation of a (see below). The first means that the clause is an element of F; the second means that for every instance of the name, the clause is a member of Δ (i.e. each instance of the clause can be used in a theory). The name is used as a way to refer to the default, as well as a way to specify which variables are significant in making instances of possible hypotheses.

Tasks appropriate for the Theorist framework

Nonmonotonic reasoning — Reasoning with Default and Generalised Knowledge

The notion of default reasoning is ubiquitous in common-sense reasoning, and is manifest in all our reasoning from inconsistent and incomplete information. The default logic of Reiter [Reiter80] formalizes the notion of default rules in a logic that uses a non-monotonic reasoning strategy to draw conclusions from defaults that are consistent with an intended model. We have shown [Poole84] how defaults can be seen as possible hypotheses to be used in a theory to explain the answers. These possible hypotheses are pieces of knowledge which can be used as long as they are consistent with everything else. This was the initial intuition behind Reiter's defaults. Note that we have translated defaults not as probability statements, or statements of "typically", but rather as things we are prepared to accept in an explanation.

Tweety the non-flying bird is the most common example of the inadequacy of monotonic reasoning from consistent facts. "birds fly" and "Tweety is a bird" suggests that "Tweety can fly," however subsequent observations like "Tweety is a penguin," and "Penguins can't fly" reveals the first inference to be too eager. While this example explains why logical consequence is alone an inadequate model of common-sense reasoning, there are many natural task domains where the same phenomenon arises.

Theorist handles this by treating the defaults that we want to use as possible hypotheses. Any answer derived from the default has the explicit dependency on the theory which we cannot show is inconsistent.

Figure 2 is a dialogue with Theorist that shows how Theorist can do the above reasoning. Lines that begin with an embolden command indicate user inputs. The syntax "n(X)" means "$\neg X$." Note that there is consistent explanation for *flies*(*tweety*) because *flies*(*tweety*) can be initially explained by the theory *birdsfly*(*tweety*), but when we add the fact that tweety is an emu, that theory is no longer consistent, as its negation can be proven. We can then no longer explain

```
Welcome to Theorist 0.1
type "help;" for help
good luck
:default birdsfly(X) flies(X) <- bird(X);
:fact n(flies(X)) <- emu(X);
:fact bird(tweety);
:explain flies(tweety);
yes
Theory: [birdsfly(tweety)]
Answer: []
:fact emu(tweety);
:explain flies(tweety);
no
:quit;
```

Figure 2 Default reasoning in Theorist 0.1

tweety flying.

Diagnosis

Most current *rule-based* diagnosis systems use knowledge of the form

observation and knowledge of situation $\rightarrow$ *problem*

to express knowledge about the potential cause of an observation. For example, the medical diagnosis system MYCIN [Buchanan84] expresses its knowledge in terms of rules of the form

symptom $\rightarrow$ *disease [CF]*

where *CF* is a *certainty factor* in the interval [-1,1] that represents a subjective evaluation of the rule's quality. The diagnosis task consists of matching rule *symptoms* and observed symptoms, accumulating the conclusions suggested by relevant rules, and ranking the conclusions by a simple arithmetic function on certainty factors.

This rule-based approach has a number of problems:

1. There is the philosophical problem of what the certainty factors mean. What does it mean for a rule to be correct? Is there a way of looking at a rule independently of the whole system?
2. There is the pragmatic problem obtaining the certainty factors. The only methodology we have is to guess, and then tune the system when unintuitive answers result. We can then never be sure that our rule base was correct, unless we have checked all combinations.

3. Such a systems produces only an assignment of certainty factors to labels. If we desire a deeper explanation of a problem, we require a theory from which the observed symptoms can be deduced. An "explanation" is a consistent grouping of possible hypotheses; the Mycin system cannot generate such groupings of hypotheses.
4. The rules above are the wrong way around: diseases result in symptoms, rather than symptoms resulting in diseases.

The rule-based representation encodes the *methodology* of the diagnostician instead of knowledge about diseases. The certainty factors are partly a record of experience (what has been previously observed) and partly a representation of belief about the relationship between symptoms and diseases.

Theorist uses an alternative formulation of the rules, viz.,

problem $\longrightarrow$ *observation*

where knowledge is expressed in terms of problems and the observations that consequently arise. For example, the medical diagnosis task would use rules of the form

disease $\longrightarrow$ *symptom*

to encode the observable symptoms of diseases. This form of representation is more appropriate for expressing text book knowledge of diseases, as it records what is known without any requirement for heuristic measures like certainty factors.

The possible hypotheses then are the diseases and possible malfunctions that we are prepared to accept in a diagnosis. We can allow for inaccurate knowledge by the use of defaults. Thus the diagnosis then becomes a theory of diseases and defaults on these diseases which then explain the observations.

The example in figure 3 shows how Theorist can be used for diagnosis. The set of *possible hypotheses* which we are prepared to accept as part of an explanation are the patient has tennis elbow, patient has dishpan hands, patient has arthritis, or patient has meningitis. The *observations* are symptoms; that some part is aching, or that the patient plays tennis. This information can be offered as observations in the request for an explanation, or by answering Theorist's questions. In figure 3, the first emboldened "explain" command asks Theorist to explain how a patient could have aching hands and an aching elbow. By using a general deduction system, the system generates the possible explanatory theories:

1) patient has dishpan hands and tennis elbow, and
2) patient has arthritis.

While attempting to construct a consistent explanation, Theorist can ask relevant questions that may help reject an explanation. This can be accomplished through application of the deduction machinery to try to prove that the theories are inconsistent. Again, in figure 3, an emboldened "askable" command informs Theorist that it may query the user about whether or not some body part is aching. In the subsequent request to explain "aching(hands) and aching(elbow)," Theorist asks the user about aching hands and elbow. The "unknown" response forces Theorist to continue it's construction of an explanation based only on the observations and

```
:fact aching(elbow) <- tennis-elbow;
:fact aching(hands) <- dishpan-hands;
:fact aching(X) <- arthritis joint(X);
:fact joint(elbow);
:fact joint(hands);
:fact joint(knee);
:fact plays-tennis <- tennis-elbow;
:default patient-has-tennis-elbow tennis-elbow;
:default patient-has-dishpan-hands dishpan-hands;
:default patient-has-arthritis arthritis;
:default patient-has-meningitis meningitis;
:askable plays-tennis;
:explain aching(elbow) aching(hands);
Is plays-tennis true? (give negative instances)
yes;
yes
Theory: [patient-has-dishpan-hands,patient-has-tennis-elbow]
Answer: []
:retry;
yes
Theory: [patient-has-arthritis]
Answer: []
:retry;
no
:askable aching(X);
:explain aching(elbow) aching(hands);
Is aching(elbow) true? (give negative instances)
unknown;
Is aching(hands) true? (give negative instances)
unknown;
yes
Theory: [patient-has-dishpan-hands,patient-has-tennis-elbow]
Answer: []
:retry;
Is aching(knee) true? (give negative instances)
no;
no
:quit;
```

Figure 3 Diagnosis in Theorist 0.1

facts. Near the end of figure 3, the first explanation for "aching hands" and "aching elbow" is "dishpan hands and tennis elbow", as in the first case. In otherwords none of the user responses made any difference in determining the first possible explanation. However, when ask to consider another possible explanation by requesting a "retry," Theorist again prompts with the question "Is aching(knee) true?" Having already received "unknown" responses about hands and elbows,

Theorist is asking about other body parts in order to attempt to refute the possible hypothesis "patient-has-arthritis." The response "no" provides the necessary information; arthritis is refuted, so no other explanations are possible. This corresponds to empirically checking a scientific theory by allowing it to make predictions, and then modifying the theory if the facts do not correspond with the predictions.

Learning as theory construction

Figure 3 shows that there may be multiple consistent theories that provide an explanation for a given set of observations. In general, one expects that the "best" explanation for the diagnostic problem is the *most specific theory* (within the constraint of having the knowledge, and the desire to discriminate between different more specific theories). That is, we would prefer an explanation like "The observations are a consequence of bronchial pneumonia" rather than a less specific explanation like "The observations are a consequence of a respiratory disorder." The motivation for preferring the most specific explanation is the desire to provide the most appropriate treatment. We [Jones85] have described a way to do a hierarchical diagnosis based on Theorist and this notion of the best theory.

If we are interested in the accumulation of rules to explain new observations, then the most specific of a number of consistent theories is not what we desire. Rather we expect the *most general theory* to be more interesting, as it presumably may account for more observations. This is precisely the intuition behind Carbonell, Michalski and Mitchell's explanation of how one recognizes improvement within the *knowledge acquisition* form of machine learning: "A person is said to have learned more if his knowledge explains a broader scope of situations, is more accurate and is better able to predict the behavior of the physical world." [Carbonell83, p. 6].

The notion of theory construction is used in several models of machine learning (e.g., [Hayes-Roth83, Lenat83, Langley83]), where the major emphasis is on strategies for generating and using *useful* hypotheses. For example, Hayes-Roth [Hayes-Roth83] concentrates on heuristics for controlling a theory building mechanism. In fact, with Theorist as a basic foundation, Hayes-Roth's five heuristics for dealing with refuted hypotheses amount to strategies for controlling the way in which Theorist backtracks (*retraction*, *avoidance*) or modifies selected hypotheses (*exclusion*, *assurance*, and *inclusion*). A cursory analysis suggests that distinguishing theory revision by hypothesis modification and theory revision by hypothesis deletion will clarify the differences that Hayes-Roth notes.

Notice that Theorist suggests nothing about where hypotheses come from or how they should be employed. This, we claim, is the essence of learning research; Theorist merely offers a foundation on which to experiment with various strategies of theory construction and maintenance, in the same way that Prolog provides a basic logical foundation for programming reasoning strategies.

User modelling as theory maintenance

A fundamental problem in the development of more sophisticated computer-mediated learning is the maintenance and use of user models [Sleeman82]. For example, a program that is helping a child to learn arithmetic will only be able to

deal with that child's specific problems if the program maintains a model of the child's performance. Current approaches to user modelling (e.g., [Brown82, Goldstein82]) coincide in their emphasis on the construction and maintenance of *evolving* models of a user. The essence of intelligent computer-aided instruction is to improve a user's understanding of a subject area by revealing misconceptions that exist in that user's current model of the domain. The foundation of both Goldstein's WUSOR-I program and Brown et alias' SOPHIE programs is the maintenance of a "theory" that is consistent with what has been observed by the program, or understood by the user.

For example, SOPHIE uses a theory construction procedure to show its user how to debug a faulty electronic circuit. An important aspect of SOPHIE is to teach a debugging strategy by interactively refining a theory of the faulty component. The strategy, briefly summarized, is to refine a model of the faulty circuit by proposing hypotheses and then verifying hypotheses by making measurements. Hypothesis formation in SOPHIE proceeds in three stages: (1) examine each observation and propose a list of hypotheses which explain it; (2) simulate each hypothesized fault to verify that it accounts for the circuit measurements; and (3) instantiate partially specified hypotheses and test their consistency.

The Theorist framework simplifies this process: measurements are the *observations* to be explained, *facts* correspond to general knowledge about the circuit, and *possible hypotheses* correspond to potential explanations for the observed behaviour. Again, as in learning by theory construction, note that Theorist does not provide any expertise about forming hypotheses, nor does it suggest any method for preferring one consistent explanation over another. However, it does provide a logical system for constructing consistent explanations regardless of domain, which frees the ICAI system designer to concentrate on domain issues (e.g., the expertise required for hypothesis generation and selection), pedagogical issues (e.g., the management of user interaction and participation), and the development of concepts for relating different explanations that a user might believe (e.g., of one explanation being a refinement of another).

Choices in mundane tasks

Even government bureaucracy relies on a style of reasoning that requires one to pursue multiple paths of reasoning based on different original hypotheses. For example, Revenue Canada's 1985 Instalment guide for individuals explains

> You may determine the amount of your instalment payments of Income Tax for 1985 by one of three methods...choose the method that will result in the lowest possible quarterly remittance [Canada85, p. 1].

In this situation, the collection of *possible hypotheses* include three different methods for determining the quarterly income tax payments: 1) base the instalment on the 1984 taxes and estimated deductions in 1985, and none of the following apply: you reside outside of Canada, you received income from a source outside of Canada, you have rental or self-employment income, you have forward averaging income, you have a share purchase credit or a scientific research tax credit, you have claimed a business investment tax credit or employment tax credit. 2) base the instalment on the 1984 taxes and estimated deductions in 1985, and at least one of

the above exceptions apply; 3) base the instalment on the 1985 estimated income and deductions.

The *facts* include all the regular rules about how income and deductions are manipulated to produce an estimate of the instalment payment. *Observations* would include income and deduction data from 1984 and 1985. Clearly one would like *Theorist* to construct the three possible explanations, and select the one in which the instalment estimate was least.

Representation and reasoning in Theorist

Given the semantics of Theorist we want to now show how it can be implemented, based on a Prolog-technology deduction system. We will show how we can implement this in Prolog, but plan to implement it at the same level as Prolog, using the advances being made in logic programming implementation techniques.

Extending Horn clauses to full first order logic

Unfortunately we cannot directly use a Prolog implementation as Horn clause logic is alone inadequate for the deductive basis of Theorist. Semantically, this is because Horn clause theories are always consistent, i.e., there is always a true interpretation for any set of Horn clauses. Pragmatically this is because Horn clause theories can never derive the negation of a formula; one can never show a Horn clause theory to be incorrect. For this, we need to add negation. With negation, we also get disjunction, as $x \subset \neg y$ is the same as $x \vee y$.

The resulting representation language is simply the full first order form of clausal logic. The current Theorist prototype will accept, as assertions, any formula of the form

$$L_1 \vee L_2 \vee \cdots \vee L_n \leftarrow L_{n+1} \wedge L_{n+2} \wedge \cdots \wedge L_{n+m}$$

where each L_i is an arbitrary literal. While we have chosen this simple language as the basis for representation in Theorist, we recognize the advantage of providing representation structures that are suggestive of the manner in which a user conceives his application domain. Therefore, an extension currently being considered is the addition of descriptions [Goebel85]. This extension will provide the Theorist user with the ability to refer to individuals and set by their properties, and to assert relations on such descriptions.

Reasoning as the construction of consistent theories

The semantics of Theorist allow us to directly implement a deduction system by treating validity as derivability. The construction of consistent explanations is done in two steps. Each step uses a first order theorem prover, and so each step is only semi-decidable. The combination makes explainability completely undecidable. This corresponds to never being really able to show our beliefs are correct and consistent, or that scientific theories can be shown to be the correct theory.

As explained above, each user "explain G" command initiates a procedure that attempts to find a consistent theory to explain the observations G. Mirroring the semantics given above, we seek a procedure that can select a subset D of Δ

such that

$F \cup D \vdash G$ and
$F \cup D$ is consistent

The first step, and first use of the first order proof procedure, is to try to prove G, using elements of F and Δ as axioms. The strategy in step one is to use a goal-directed, backward theorem-prover that will generate subgoals that can be used to select relevant hypotheses from the set Δ. We then make D those elements of Δ actually used in the proof. As we have previously noted, the intelligent selection of hypotheses is a major problem (below we describe a simple strategy that is similar to the way in which Prolog considers assertions in the construction of an SLD proof tree).

The second step also requires the use of a first order proof procedure. To verify that a subset D of Δ together with the facts is consistent, we attempt to show that

$$F \cup D \nvdash \neg d$$

for each $d \in D$. We assume that the set F of facts is consistent (i.e., the intended interpretation is a model for F), so that we need only verify the consistency of each hypothesis used in the explanation. Notice that the second step requires the use of a *complete* first order proof procedure, and that, like the previous step, the computation is semidecidable. Note, however, that if no defaults were used, and the knowledge was in the form of Horn clauses, then this systems defaults to exactly Prolog.

Implementing a Theorist prototype in Prolog

Here we show how we can construct a full first order clausal theorem-prover in Prolog.

The extension of Prolog to a first order clausal theorem-prover is equivalent to Loveland's development of the *MESON* proof procedure, as based on the *problem reduction* problem solving strategy [Loveland78]. Loveland's development of the resolution-based MESON procedure represents his logical reconstruction of the goal-driven problem solving strategy that reasons by reducing a given goal to a set of "simpler" subgoals. Loveland's departure point is the use of informal rules of the form

$$goal \leftarrow subgoal_1 \,\&\, subgoal_2 \,\&\, \cdots subgoal_n$$

which, of course have the format of Horn clauses: at most one positive literal when in clause form (recall that $L \leftarrow L_1 \,\&\, L_2 \,\&\, \cdots L_n$ is equivalent to $L \vee \neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_n$)

A full first order theorem-prover based on Prolog has been constructed by Umrigar and Pitchumani [Umrigar85] and is based on Loveland's description of the MESON procedure. The implementation here is similar, except that we have not implemented any cycle check, nor the mechanism that will produce indefinite answers. We note here that the cycle check used in Umrigar and Pitchumani's F-Prolog system is not correct. See [Poole85a] for a description of the problems with

some of the suggested techniques for eliminating loops in Prolog proofs.

The fundamental idea behind the implementation of a first order theorem-prover in Prolog can be understood by considering the following example. Suppose that we are given the propositional clauses

$$a \vee b$$
$$c \leftarrow a$$
$$c \leftarrow b$$

and the goal c. Even if we could state this as Horn clauses, Prolog's response to the goal is "no"; there is no SLD derivation of c, even though it is easy to see that c is valid, given that one of a or b is true. As Loveland explains (see [Loveland78, pps. 341ff.]), what is lacking here is negation as required to express the contrapositive form of the assertions given above. If we augment the above assertions with their *general contrapositives* we get

$$a \vee b$$
$$a \leftarrow \neg b$$
$$b \leftarrow \neg a$$
$$c \leftarrow a$$
$$\neg a \leftarrow \neg c$$
$$c \leftarrow b$$
$$\neg b \leftarrow \neg c$$

The general backchaining strategy will now produce the proof branch

$$c \text{ --- } (c \leftarrow a) \text{ --- } a \text{ --- } (a \leftarrow \neg b) \text{ --- } \neg b \text{ --- } (\neg b \leftarrow \neg c) \text{ --- } \neg c$$

which can be marked as successful by virtue of a *reductio ad absurdum* argument. (See Loveland [Loveland78] for a full elaboration.) That is, during the construction of a proof branch, the proof terminates successfully if the negation of a new subgoal occurs further up the subtree. In otherwords, the implementation of a full first order theorem-prover in Prolog requires 1) the addition of the contrapositive forms of each assertion, and 2) an elaboration of the Prolog search procedure to look for for the negation of a newly generated subgoal anywhere earlier in the current proof branch.

Figure 4 is a listing of the current implementation of the Theorist proof system in Waterloo UNIX† Prolog Note that this implementation is only correct when there are no Skolem functions (i.e. none of the functions should be interpreted as Skolem functions) and there are no free variables in hypotheses when they are tested for consistency.

Of most interest is the definition of the proof predicate *pr*. The relation *pr(G A D1 D2)* means that the observations G can be proved, with proof tree ancestors A, and initial theory $D1$ and final theory $D2$. The first clause for *pr* implements the search of the proof tree ancestors for a contradiction, as described above. The second clause retrieves a fact from the *fact module* and attempts to use that fact to

† UNIX is a registered tradmark of Bell Laboratories.

```
% explain(G D) is true when all goals in G are explained by the facts and consistent defaults D.
        explain(G D) <-
                prall(G [] [] D);
% prall(B A D1 D2) is true when all goals in B are proved, with ancestors A, starting defaults D1 and final defaults D2.
        prall([] A D D);
        prall([G|B] A D1 D3) <-
                pr(G A D1 D2)
                prall(B A D2 D3);
% prall(B A D) is like prall(B A D1 D2), except that only previousy selected hypotheses may be used.
        prall([] A D);
        prall([G|B] A D) <-
                pr(G A D)
                prall(B A D);
% pr(G A D1 D2) is true when G is proved, with ancestors A, starting defaults D1 and resulting defaults D2.
        pr(G A D D) <-                  % search up tree for negation of current goal
                neg(G GN)
                member(GN A);
        pr(G A D1 D2) <-                % retrieve next fact for expanding proof tree
                fact(G Body)
                prall(Body [G|A] D1 D2);
        pr(G A D1 D2) <-                % get next default/hypothesis for expanding proof
                default(N G B)
                prusedef(G A N B D1 D2);
        pr(G A D D) <-
                prolog(G Module)        % G is defined in WUP Module
                prove(Module G);
        pr(G A D1 D2) <-
                askuser(G A D1 D2);
% pr(G A D) is like pr(G A D1 D2) except that only previousy selected hypotheses may be used.
        pr(G A D) <-
                neg(G GN)
                member(GN A);
        pr(G A D) <-
                fact(G Body)
                prall(Body [G|A] D);
        pr(G A D) <-
                default(N G B)
                member(G D);
        pr(G A D) <-
                prolog(Module G);
% prusedef(G A N B D1 D2) is true when G is proved with ancestors A, using default name N, body B, starting defaults
%  D1 and final defaults D2.
        prusedef(G A N B D1 D2) <-
                member(N D1)
                prall(B [G|A] D1 D2);
        prusedef(G A N B D1 D2) <-
                not(member(N D1))
                prall(B [G|A] [N|D1] D2)
                neg(G GN)
                not(pr(GN [] D1 D1));
% neg(A B) is true when A is the negated form of B.
        neg(n(A) A) <-
                ne(A n(_));
        neg(A n(A)) <-
                ne(A n(_));
```

Figure 4 A Prolog implementation of the Theorist proof system

generate a new set of subgoals. The *fact module* is a WUP module that contains a compiled version of user assertions. The compiled version is simply clause form represented as a Prolog list. The contrapositive form of each assertion is formed dynamically by matching the current goal with a literal of the compiled list form, and then returning the negated form the the remaining literals in the list as the new subgoals. The third clause uses an auxiliary relation *default(N G B)* which retrieves the next default or hypothesis from the hypothesis module with name N, head G, and body B. As with facts, hypotheses are stored in clause form, and the appropriate form is constructed as necessary; when a literal of the clause form matches the current subgoal, the appropriate body is constructed as the new set of subgoals.

The last two clauses in the definition of *pr* are for permitting the user to declare "Prolog only" predicates, and to initiate a user dialogue based on the declaration of *askable* predicates (see discussion of figure 3 above).

Not parallelism

Note that the definition of the *pr* relation in figure 4 uses the standard Prolog sequential selection of facts to attempt the backchaining construction of a proof for the goal G. As acknowledged above, a simple assertion-order selection of both facts and hypotheses is used to search for a set of hypotheses which, when combined with the facts, supports the derivation of the observations G.

The second portion of the Theorist computation, consistency verification, is initiated whenever a new default is selected by the third clause of the definition of *pr*. The relation *prusedef* will first determine if the default has been previously used, in which case its consistency has been established and the normal proof procedure can continue. Otherwise, verification of the consistency of the newly considered hypotheses must be attempted by trying to fail to prove the negation of the conclusion suggested by the hypothesis.

In general, every hypothesis considered will spawn a (potentially infinite) computation that attempts to show that the hypothesis is consistent with the current facts and the current partial theory. We speculate that these computations can be pursued in parallel; that they define *not parallelism* which is responsible for the verification of consistency while the hypothesis selection mechanism concurrently generates plausible theories whose consistency is in question.

Note that this idea of *not parallelism* can be seen as equivalent to the idea of using a scientific theory before it has been fully tested, and then fixing up the theory when errors have been found. It also allows us to have the idea of having a set of beliefs which may be inconsistent, but still being able to use these beliefs, and fixing them up when an inconsistency is detected. All of these are consistent with the idea we used before, as a search strategy, and fit in with our logic.

Status and conclusions

Here we have presented a simple, yet very powerful methodology which unified many disparate areas of Artificial Intelligence. This is just the idea, that rather than just doing deduction from our knowledge, we should build "scientific" theories which can explain the results. A very naive implementation was presented which

allows us to play with these ideas. This implementation is a strict extension to Pure Prolog, in the sense that if all of the knowledge is in Horn clauses, and no defaults were used, then we get just Prolog.

This approach seems to ask more questions than it answers. It gives us a new way to approach old problems. For example, how we claim learning research should proceed, is to first ask the question *what does it mean to have learnt something?* We claim the answer to this is to *build a better theory of the world.* So to do research into learning, we claim that we should look at the two issues of when is one theory better than another, and how can we effectively compute the best theory for special cases, or maintain a good theory for the general case. The notion of a best theory will change depending on whether we are using defaults [Poole85b] or diagnosis [Jones85] and acknowledge that much research still needs to be done.

Acknowledgements

We have fruitful discussions about the concept of Theorist, and theory formation in general, with Ray Reiter, Russ Greiner, Michael Genesereth, Bob Hadley, Jim Delgrande, and the other members of the Logic Programming and AI Group of the University of Waterloo: Robin Cohen, Maarten van Emden, and Marlene Jones. This research has been supported by National Sciences and Engineering Research Grant No. A0894.

References

[Brown82] J.S. Brown, R.R. Burton, and J. de Kleer (1982), Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III, *Intelligent Tutoring Systems*, J.S. Brown (eds.), Academic Press, New York, 227-282.

[Buchanan84] B.G. Buchanan and E.H. Shortliffe (1984, eds.), *Rule-Based Expert Systems The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, Massachusetts.

[Canada85] Revenue Canada (1985), *1985 Instalment Guide for Individuals*, Revenue Canada Taxation Centre, Ottawa, Ontario.

[Carbonell83] J.G. Carbonell, R.S. Michalski, and T.M Mitchell (1983), An overview of machine learning, *Machine Learning An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M Mitchell (eds.), Tioga, Palo Alto, California, 3-23.

[Cheng84] M.H.M. Cheng (1984), The design and implementation of the Waterloo Unix Prolog environment, M.Math thesis dissertation, Department of Computer Science, University of Waterloo, September, 114 [ICR Report 26; Department of Computer Science Technical Report CS-84-47].

[Goebel85] R. Goebel (1985), Interpreting descriptions in a Prolog-based knowledge representation system, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, August 16-18, UCLA, Los Angeles, Cali-

fornia, 711-716.

[Goldstein82] I.P. Goldstein and R.R. Burton (1982), The genetic graph: a representation for the evolution of procedural knowledge, *Intelligent Tutoring Systems*, J.S. Brown (eds.), Academic Press, New York, 51-77.

[Hayes-Roth83] F. Hayes-Roth (1983), Using proofs and refutations to learn from experience, *Machine Learning An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M Mitchell (eds.), Tioga, Palo Alto, California, 221-240.

[Jones85] M. Jones and D. Poole (1985), An expert system for educational diagnosis based on default logic, *Proceedings of the Fifth International Workshop on Expert Systems and their Applications*, May 13-15, Palais des Papes, Avignon, France, 573-583.

[Langley83] P. Langley, G.L. Bradshaw, and H.A. Simon (1983), Rediscovering chemistry with the Bacon system, *Machine Learning An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M Mitchell (eds.), Tioga, Palo Alto, California, 307-329.

[Lenat83] D.B. Lenat (1983), The role of heuristics in learning by discovery: three case studies, *Machine Learning An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M Mitchell (eds.), Tioga, Palo Alto, California, 243-306.

[Loveland78] D.W. Loveland (1978), *Automated theorem proving: a logical basis*, North-Holland, Amsterdam, The Netherlands.

[Poole84] D.L. Poole (1984), A logical system for default reasoning, *Proceedings of the AAAI Workshop on nonmonotonic reasoning*, October 17-20, New Phaltz, New York, 373-384.

[Poole85a] D.L. Poole and R. Goebel (1985), On eliminating loops in Prolog, *ACM SIGPLAN Notices* **20**(8), 38-41.

[Poole85b] D.L. Poole (1985), On The Comparison of Theories: Preferring the Most Specific Explanation, *Proceeding of the Ninth International Joint Conference on Artificial Intelligence*, August 16-18, UCLA, Los Angeles, California, 144-147.

[Popper58] K.R. Popper (1958), *The Logic of Scientific Discovery*, Harper & Row, New York.

[Quine78] W.V.O. Quine and J.S. Ullian (1978), *The Web of Belief*, Random House, New York.

[Reiter80] R. Reiter (1980), A logic for default reasoning, *Artificial Intelligence* **13**(1&2), 81-132.

[Sleeman82] D. Sleeman and J.S. Brown (1982, eds.), *Intelligent Tutoring Systems*, Academic Press, New York.

[Umrigar85] Z.D. Umrigar and V. Pitchumani (1985), An experiment in programming with full first-order logic, *IEEE 1985 Symposium on Logic Programming*, July 15-18, Boston, Massachusetts, 40-47.