Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning

Randy Goebel
Koichi Furukawa
David Poole

Research Report CS-85-50
December 1985
Revised March 1986

Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning

Randy Goebel†
Koichi Furukawa‡
David Poole†

†Logic Programming and AI Group
Department of Computer Science
University of Waterloo
Waterloo, CANADA N2L 3G1

‡First Laboratory
Institute for New Generation Computing
21F, Mita Kokusai Bldg, Minato-Ku
Tokyo, 108 JAPAN

Abstract

If one desires that an automatic theory formation program detect inconsistency in a set of hypotheses, the Horn clause logic of Prolog is unsuitable as no contradiction is derivable. Full first order logic provides a suitably expressive alternative, but then requires a full first order theorem-prover as the basic theory construction mechanism. Here we present an alternative for augmenting definite clauses with the power to express potentially inconsistent scientific theories. The alternative is based on a partitioning of definite clauses into two categories: ordinary assertions, and integrity constraints. This classification provides the basis for a simple theory formation program. We here describe such a theory formation system based on Prolog, and show how it provides an interesting reformulation of rule-based diagnosis systems like MYCIN.

1. Introduction

In general, the automatic formation of scientific theories seems to require formalization in a language in which it is possible to derive the negation of a formula [Poole86b]. This is because we are interested in systems that deduce contradictions to reject inconsistent theories, rather than those that reject theories by being told of an inconsistency with respect to a complete description of the intended interpretation. The latter view of theory formation is appropriate when considering the problem of synthesising logic programs from example facts [Shapiro82]. The former, however, requires that the system detect inconsistent theories syntactically, if possible, and maintain a consistent theory of a partial description of the intended interpretation.

Horn clause logic, in particular the definite clause subset comprising Prolog assertions, is not capable of expressing that some formula f is not true. Furthermore, Prolog's proof procedure is incapable of deriving that negation of a formula, except by negation-as-failure [Clark78].

As a representation language for expressing formulas from which theories can be formed, Prolog's definite clauses are unsuitable, as every set of definite clauses is consistent. Poole et al. [Poole86b] have demonstrated a program called Theorist that shows how first order logic can provide the foundation for defining a system for automated theory formation. That program uses full first order logic, and requires a full first order theorem-prover as the basic reasoning mechanism (cf. [Umrigar85]).

Here we present an alternative for augmenting definite clauses with the power to express potentially inconsistent scientific theories. Our motivation is to investigate the problem solving power of a system that is simpler than Theorist, but that uses the same idea of deduction-based theory formation. The alternative is an extension to the definite clause assertion language of Prolog, and is based on a partitioning of definite clauses into two categories: ordinary assertions, and integrity constraints. This classification provides the basis for the development of a simple theory formation program based on Prolog.

In section 2 we summarize the method for defining a theory formation program in terms of a representation and reasoning system based on first order logic. Section 3 describes a partitioning of definite clause knowledge bases that creates a distinction between tacit and necessary assertions. The concept of necessary assertion is based on the integrity constraints used in logic data bases (e.g., [Kohli83, Miyachi84, Goebel85a, Goebel85b) These necessary assertions provide the basis for accepting some collections of formulas as consistent and rejecting others as inconsistent. This, in turn, provides the necessary expressive power to specify a theory formation program like Theorist, but based on a simpler deductive mechanism. This version of Theorist is called *Theorist-S*. In section 4, Theorist-S is used to specify a simple rule-based diagnosis system. A comparison with MYCIN shows how MYCIN's rules can be viewed as Theorist-S assertions. While other versions of Prolog in MYCIN have already been described (e.g., [Hammond82]), our explanation differs in that we demonstrate how the rules of MYCIN incorporate both object and meta level logical assertions, and that the separation of these kinds of information is useful in developing new diagnosis systems based on theory formation. In addition, we show how the separated object and meta level MYCIN rule knowledge can be automatically reconstructed by using the partial evaluation technique of Takeuchi et al. [Takeuchi85]

Finally, we suggest a method for dealing with a limited form of "certainty factor," based on an objective measure that records the numbers of tacit and necessary facts used to explain the observations. We conclude with a list of ways in which this work should be further pursued.

2. Using deduction to build scientific theories

Our definition of "scientific theory" derives from Poole et al. [Poole86b], which is based on Popper's logic of scientific discovery [Popper58]. Like Theorist, the framework of Theorist-S consists of the basic components shown in fig. 1.

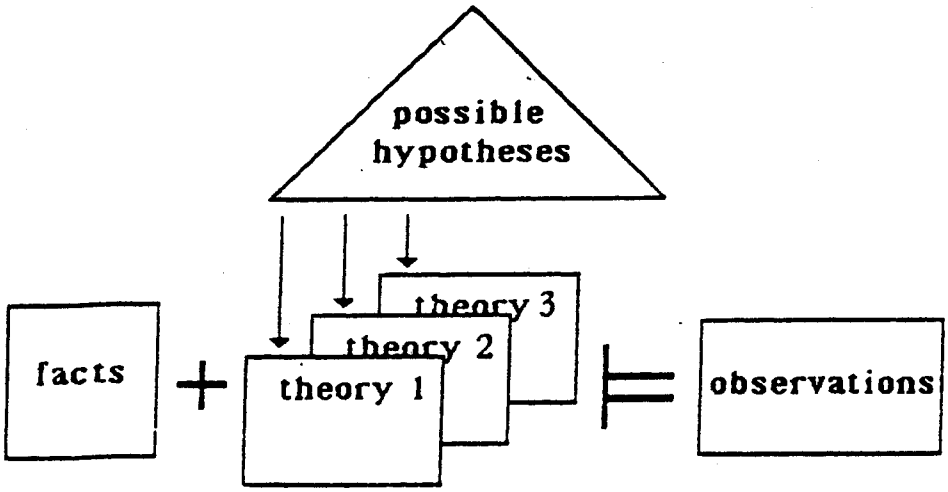

Figure 1 Framework of Theorist and Theorist-S

The knowledge of the system is represented as a set of formulas of a logical language, say L, which is divided into four categories. The *facts* are a set of formulas of L that have the user's intended problem domain as a model. The *possible hypotheses* are a set of formulas of L, instances of which may be required to augment the facts in order to explain the *observations* — another set of formulas of L representing domain observations for which an explanation is desired. The relation $\vdash$ denotes the provability relation for the logic of language L. A *scientific theory* T (e.g., $T1$, $T2$, $T3$ in fig. 1) is an *explanation* for a set of observations O if and only if $\forall t \in T$, t is an instance of h, where $h \in$ possible hypotheses, and

$$T \bigcup Facts \vdash O \tag{1}$$

$$\forall t \in T, T \bigcup Facts \nvdash \neg t. \tag{2}$$

Intuitively, T is a set of consistent hypotheses that, together with the facts, support the deduction of the observations.

As has been elsewhere suggested (e.g., [Meltzer73]), Theorist uses deduction to *attempt* to show the consistency of a set of formulas augmented with unverified hypotheses. If this consistency is established and the set of formulas can be shown to derive the observations, that set of consistent hypotheses is called a theory that

explains the observations.

The word "attempt" is most important here, for if the language L has as least the expressive power of first order predicate language with dyadic predicate symbols then the relationship described by formula (2) cannot, in general, be effectively determined. Furthermore, the proof procedure denoted by the symbol E must be complete so that, in cases when the relationship is determined, one can conclude the consistency of the hypothesis in question. Note that appealing to an oracle with complete knowledge of the intended interpretation does not make theory formation decidable. For example, without heuristic control, Shapiro's *model inference system* is undecidable because it has a step requiring the proof of an arbitrary formula [Shapiro81, Shapiro82].

These theoretical difficulties not withstanding, Poole et al. argue that the Theorist framework should be considered as a unifying framework for various reasoning paradigms used in artificial intelligence research. For example, default reasoning and diagnostic reasoning can be simply reformulated in the Theorist framework [Poole86a]. The potential value of Theorist is based on identifying classes of theories which can be effectively formed, and which can be applied to interesting problem domains. For example, Reiter speculates that most diagnosis problems are propositional hence the consistency computation is effective.†

Note that Theorist does not suggest any particular method for selecting hypotheses to augment the current facts or suggest any method for dynamically inducing relevant hypotheses. However, is does provide a logic programming system in which various strategies for automatic theory formation can be investigated and developed.

3. Falsifiability and the expressive power of L

A final qualification on the expressive power of L must be made, in order that theories being formed are *falsifiable*. This requirement is based on Popper's thesis that a scientific theory must not only be consistent, but must be potentially falsifiable, in order to be able to distinguish between all subsequent observations that might be made (e.g., see [Popper58], pps. 91-92). In the Theorist program of Poole et al., the property of falsifiability is viewed as the ability to deduce the negation of observed facts, e.g., an observation o will falsify a theory $T \bigcup Facts$ if $T \bigcup Facts \vdash \neg o$. The necessary expressive power is obtained by choosing L as a full first order predicate language, in which case the proof procedure named as $\vdash$ is a complete first order predicate calculus theorem-prover.

The prototype implementation of Theorist reported in [Poole86b] chose L to be the full clausal form of the first order predicate calculus language, and implements $\vdash$ in Prolog, according to Loveland's MESON procedure [Umrigar85, Loveland78] Our simplified version of Theorist, called Theorist-S, is strictly weaker than Theorist, but does not require the implementation of a full clausal theorem-prover for $\vdash$. In addition, the representation language L of Theorist-S is based on a simple extension of definite clauses, rather than the full clausal form of first order

† personal communication

predicate calculus.

4. Definite clauses as integrity constraints

If we select the definite clause subset of the first order predicate logic as our representation language L of Theorist-S, we face an immediate problem. Every set of definite clauses is consistent, and therefore any hypothesis in the form of a new definite clause can be added to an existing clause set without affecting that set's consistency. In fact, with L chosen as the set of definite clauses, the initial facts F can be merely augmented with the observations O to create a consistent theory $F \bigcup O$ that explains those observations.

Required here is some method for denying "theory membership" to certain hypotheses. In other words we require some extension to L, together with the related semantic and proof theoretic extensions, that allows one to express a fact that may be inconsistent with a number of potential hypotheses. Here the notion of *integrity constraint* is appropriate, especially as used by Miyachi et al [Miyachi84]. and Goebel [Goebel85a, Goebel85b]. These authors extend the usual definition of a definite clause database by classifying certain clauses as integrity constraints, and then using those constraints to verify the consistency of new assertions at database update time. This use of constraints is similar to that of Kohli and Minker who show how integrity constraints can be dynamically used to prune the search space of an SLD proof procedure [Kohli83].

This notion of integrity constraint is particularly attractive because it is so simple. First, a subset of database clauses is syntactically distinguished as integrity constraints. For example, since it is common to write definite clauses in the form

$$<\!consequent\!> \subset <\!antecedent\!> \tag{3}$$

we might distinguish constraints by writing them as

$$<\!antecedent\!> \supset <\!consequent\!>. \tag{4}$$

Though the logical semantics of schemata (3) and (4) are identical we can use the syntactic distinction to identify formulas of the form (4) as assertability conditions as new assertions. In other words, given a database *KB* and a new assertion α, all consequents of formulas of the form (4) whose antecedents unify with α define a conjunction whose derivability determines α's assertability. For example, one could not assert "*has* (*Ohki* ,*cold*)" if the constraint "$\forall x$ *has* (*x* ,*cold*)$\supset$*has* (*x* ,*sneezing*)" was asserted and "*has* (*Ohki* ,*sneezing*)" was not derivable in the current *KB*.

In the theory formation framework, integrity constraints such as these provide a simple method for denying theory membership to a hypothesis. The consistency of a theory will be constrained by the user's specification of integrity constraints.

5. Diagnosis as theory formation

The problem of diagnosing malfunctions in a complex system can be viewed as a special case of automatic theory formation (e.g., [Pople77, Brown82, Genesereth85]). For example, the medical diagnosis problem can be formulated as the problem of determining which of a number of possible diseases "best" account for a set of observed symptoms. In terms of theory formation, the facts comprise statements that represent relationships between diseases and their symptoms, and possible hypotheses include all those diseases that are acceptable as explanations for a given set of symptoms.

5.1. A simple diagnosis system based on Theorist-S

The facts of Theorist-S are specified as a collection of simple formulas of the form

$<disease> \supset <symptoms>$.

In the diagnosis system described here, relationships between diseases and symptoms are further classified as "tacit" or "necessary" in order to distinguish symptoms which may appear and those which must appear. These necessary relationships are treated as the integrity constraints which constrain the admissible hypotheses. For example, the formulas of fig. 2 represent naive knowledge about the symptoms of several common afflictions.

---

1. $\forall x$ has(x, cold) $\supset$ has(x, sneezing)
2. $\forall x$ has(x, cold) $\supset$ has(x, coughing)
3. $\forall x$ has(x, cold) $\supset$ has(x, runny-nose)*

4. $\forall x$ has(x, hayfever) $\supset$ has(x, runny-nose)
5. $\forall x$ has(x, hayfever) $\supset$ has(x, sneezing)
6. $\forall x$ has(x, hayfever) $\supset$ has(x, watery-eyes)*

7. $\forall x$ has(x, influenza) $\supset$ has(x, diarrheoa)
8. $\forall x$ has(x, influenza) $\supset$ has(x, headache)
9. $\forall x$ has(x, influenza) $\supset$ has(x, fever)*

10. $\forall x$ has(x, cold)
11. $\forall x$ has(x, hayfever)
12. $\forall x$ has(x, influenza)

Figure 2 Theorist-S naive diagnosis knowledge base

The asterisks distinguish integrity constraints, which are facts that describe "necessary" symptoms of the disease. Note that necessary symptoms are *not* pathognomonic (cf. [Pople77]) because they are not sufficient conditions for diagnosing the affliction in question. Intuitively, the facts of fig. 2 assert that you *may* cough if you have a cold, but that you will certainly have a runny-nose.

Note that tacit and necessary assertions are distinguished in that assertions with an asterisk *always* imply their consequences while the tacit assertions do not. One possible interpretation is that the meaning of "$\supset$" is modified by "*"; intuitively, the computational interpretation is that "*" assertions must be verified while others need not be. This distinction can be used, for example, as a heuristic to distinguish preferred theories. Another view is that only the "*" assertions are facts, and that all others are default assumptions. This view, which combines defaults and diagnosis, is elaborated elsewhere [Poole86a].

Hypotheses are written as universally quantified statements, for example as in formulas 10, 11 and 12 of fig. 2. Instances of the these hypotheses will be used to form an explanation of some observed symptoms, for example with $x = Fred$, the hypothesis instance $has(Fred, hayfever)$ might explain Fred's watery-eyes and runny-nose.

The algorithm for computing an explanation of the observed symptoms is based on Prolog's depth-first backtracking algorithm for SLD resolution [Lloyd84]. In general, one views the facts and hypotheses as a pure Prolog program and the list of observed symptoms as the query to be derived. For example, Theorist-S will explain Ohki's runny nose and sneezing if the facts and possible hypotheses in fig. 2 are viewed as a pure Prolog program and we pose the query

$$?has(Ohki, runny\text{–}nose) \wedge has(Ohki, sneezing) \qquad (5)$$

When an instance of a possible hypothesis is used in a branch of the SLD proof tree for the goal (5), that instance is recorded as part of the explanation for the observed symptoms. An explanation for a set of symptoms is the union of the instances of possible hypotheses that where used in an SLD derivation of the symptoms. This strategy is the same one suggested by Shapiro [Shapiro82, p. 160], and used by Finger and Genesereth's RESIDUE system [Finger85] For example, fig. 3 is a diagram of the SLD proof tree for the goal (5), using the facts and possible hypotheses given in fig. 2. We can extract the four possible theories or explanations from the complete SLD tree; there is one for each successful SLD branch:

I: {has(Ohki, cold)}
II: {has(Ohki, cold), has(Ohki, hay-fever)}
III: {has(Ohki, hay-fever), has(Ohki, cold)}
IV: {has(Ohki, hay-fever)}

Each of the above explanations, together with the facts, support the derivation of the observed symptoms. However, we have not yet tested the consistency of the explanations. In Theorist-S the constraints (indicated by * in fig. 2) specify those symptoms that must be present in order to consistently assume the possible hypothesis. For example, the assertion

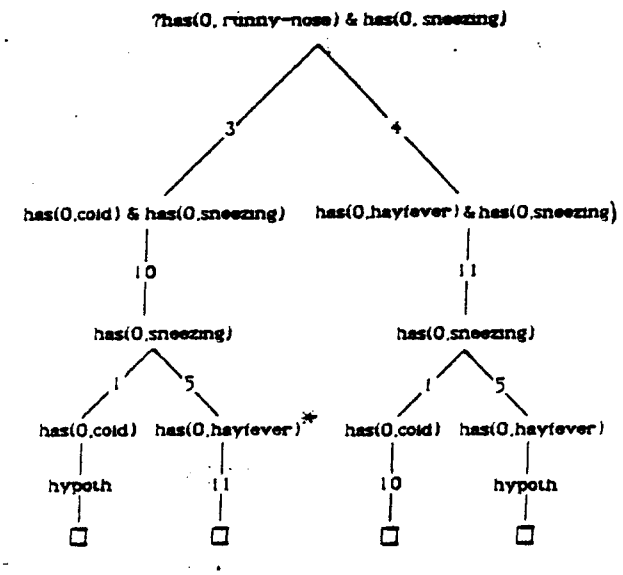

Figure 3 SLD proof tree for Theorist-S diagnosis

$$\forall x \; has(x, cold) \supset has(x, runny-nose)^*$$

declares that $has(\alpha, cold)$ and $\neg has(\alpha, runny-nose)$ are incompatible for any individual α. In other words, the possible hypothesis $has(\alpha, cold)$ is acceptable only if it can be verified that $has(\alpha, runny-nose)$.

In the current Theorist-S prototype, new instances of possible hypotheses are verified as consistent as they arise in the SLD proof tree. For example, when the possible hypothesis instance $has(Ohki, hay-fever)$ is first considered at the point labeled (*) in fig. 3, all relevant constraints are accumulated using the algorithm specified by Miyachi et al. and Goebel. The only relevant constraint here is given by assertion (6) in fig. 2, which requires that the observation $has(Ohki, water-eyes)$ be verified. If the needed observation is not contained in the original list of observations, Theorist-S consults the user by interactively asking the question "Does Ohki have watery eyes?" If the response is "yes," the hypothesis instance is verified, and the SLD derivation continues. If the response is anything other than "yes," Theorist-S assumes the answer is "no" and rejects the hypothesis instance as inconsistent with the observations. Note that the answer "unknown" is not possible — Theorist-S assumes that the user can either answer the question or perform some experiment that will decide what the answer is. As noted by others (e.g., [Shapiro81), this interaction corresponds to conducting an experiment.

For the example query (5) above, this hypothesis verification technique would reject explanations I, II, and IV, assuming that the user answered "no" to the question about Ohki's watery eyes. Therefore the explanation that Ohki has a cold is the only explanation.

It is important to realize that this incremental verification of hypothesis consistency is, in general, inadequate. As the incremental approach is inherently sequential, it is sensitive to the order in which observations are explained. For example, consider the following example knowledge base:

1. A^*
2. B^*
3. $A \supset \neg B$

If we attempt to explain $A \wedge B$, we first assume A using 1, and since there are no axioms to disprove A, we conclude it is consistent. Then, while assuming A, we are further to required to assume B using 2. However, in this case, the previous assumption A allows the consistency test $?\neg B$ to succeed, which contradicts our assumption of B and prevents us from constructing a consistent explanation. However, an explanation of $B \wedge A$ is possible: the initial assumption of B using 2 is consistent and there is no clause whose head matches A, so we conclude the second necessary assumption is consistent as well. This gives the theory A, B as explanation, which is inconsistent. In Theorist proper [Poole86b], the first order clausal theorem prover has avaliable the contrapositive forms of all clauses. Its proof procedure can use the contrapositive of 3 in order to verify the inconsistency at the first opportunity, i.e., when A is first assumed.

5.2. Expert rules versus medical knowledge

On the surface, the knowledge encoded in rule-based expert systems like MYCIN appear to be wrong way around, i.e., they have the form

$$symptoms \supset disease \tag{6}$$

while Theorist and Theorist-S rules have the form

$$disease \supset symptoms \tag{7}$$

However, the knowledge *embedded* in MYCIN rules have the form of schema (7), as can be seen by viewing schema (6) as a *meta rule* that says "if you observe *symptoms*, then you should consider the hypothesis *disease*." This meta language expression embeds rules of the form (7) into a control framework that simplifies the description and implementation of the MYCIN production rule interpreter, at the expense of complicating the expression of knowledge about the relationship between diseases and symptoms.

The view that MYCIN combines meta and object level knowledge in its rules is further supported by noting that MYCIN is claimed to be a backward reasoning system [Buchanan84, p. 71ff.] — in fact the rule interpreter for MYCIN does *backward reasoning* on rules of the form given in schema (6), which amounts to forward reasoning on rules of the form given in schema (7).

The advantage of expressing rules in the form suggested by schema (7), is that there is *no a priori* requirement for any "expert" level control knowledge to be combined with the basic statement of relationships amongst symptoms and diseases. The theory formation procedure works on the object level knowledge expressed in the natural way (i.e., diseases causes symptoms). This has the following consequences. First, this rule form completely decouples expert experience about how to use rules, and the rules themselves. Expert knowledge like "if symptom A is observed then disease B is likely" is now viewed as a meta language assertion about the usefulness (or, in some cases, statistical appropriateness) of the object level rule (cf. [Bowen85]).

This detachment of meta and object level knowledge provides for the expression of general as well as specific meta level reasoning strategies. As Sterling suggests [Sterling84], the meta level is the appropriate level for specifying problem solving strategies. In fact, the rules of MYCIN are really *ground facts* of the logical meta language; the general predicates of the meta level axiomatization of Theorist provide an example of the concepts that should be used to express meta level knowledge or problem-solving strategies, e.g., the concepts of *fact*, *hypothesis*, *explanation*, *constraint*, and *consistent*.

Second, the meta and object decoupling means that general purpose interpreters like Theorist-S are potentially less efficient than an equivalent MYCIN-like interpreter and rule knowledge base combination. In fact the Theorist-S system uses Prolog to interpret the definition of Theorist-S, which then interprets the object level assertions of a Theorist-S application. This disadvantage, however, obviates the need for different kinds of rules (cf. MYCIN's antecedent rule [Buchanan84]). For example, Theorist-S's facts and constraints are syntactically identical — they are distinguished by the problem-solving strategy specified at the meta level.

5.3. Using partial evaluation to combine methods and knowledge

The potential inefficiency of the multi-level interpretation of Theorist-S (and all systems with similar structure) can be addressed in at least two ways. One is to construct an efficient meta level interpreter directly in a procedural language. This amounts to compilation of the meta level problem-solving strategy, which makes that strategy more efficient but more difficult to understand and modify. Since this alternative negates one of the claimed advantages of the decoupling of meta and object level knowledge, an attractive alternative for improving efficiency is partial compilation, or *partial evaluation* [Kahn84, Takeuchi85]. This technique retains most of the advantages of the meta/object decoupling, while increasing efficiency for instances of meta interpreter and object knowledge base pairs.

Returning to the claimed difference between rules of the form (6) and (7), given above, partial evaluation shows the difference to be negligible. For example, consider the MYCIN rule of fig. 4. This rule asserts that if you have the observations about gramstain, morphology and metabolism, then you can conclude that the identity of the organism is bacteriodes. Ignoring the certainty facts for the moment, we can express the information in the MYCIN rule of fig. 4 as

```
PREMISE: ($AND
     (SAME CNTXT GRAM GRAMNEG)
     (SAME CNTXT MORPH ROD)
     (SAME CNTXT AIR ANAEROBIC))
ACTION: (CONCLUDE CNTXT IDENTITY BACTEROIDES TALLY .6)
```

Figure 4 An example MYCIN rule

$$\forall x \; identity(x, bacteriodes) \supset gramstain(x, negative) \wedge morphology(x, rod) \wedge metabolism(x, anaerobic) \tag{8}$$

which asserts the relationship between the malfunction and symptoms in the form suggested by the schema (6) above. The enclosing "premise/action" directive can be viewed as a more general problem-solving strategy at the meta level, e.g.,

$$\forall identity, symptoms \; establish(identity) \subset fact(identity \supset symptoms) \; verify(symptoms). \tag{9}$$

In fact, Takeuchi and Furukawa's partial evaluation algorithm will take an appropriate form of (8) and (9) and use a program transformation technique to produce the rule

$$establish(identity(x, bacteriodes)) \subset verify(gramstain(x, negative) \wedge morphology(x, rod) \wedge metabolism(x, anaerobic))$$

which has the same form as the original MYCIN rule of fig. 4.

5.4. Preferring one diagnosis over another

We earlier noted that the '*' annotation on formulas of fig. 2 have at least two possible interpretations, both of which alter their interpretation as formulas of the first order predicate calculus. One possible interpretation retains the view of all rules as Theorist-S *facts*, and interprets unmarked formulas as those having a lower probability than those carrying the '*' annotation. The apparent discrepancy between generalizations which are always true and those which are sometimes true is then explained by associating a probability value less than one to unmarked formulas. This results in a logic programming system whose formal properties can be specified in a way similar to Shapiro's logic programming with uncertainties [Shapiro83]. Furthermore, the choice of a function or functions for determining

probabilistic entailment can be based on Nilsson's description of probabilistic logic [Nilsson84].

Another possible interpretation of the '*' annotation treats only the marked formulas as *facts* and the unmarked formulas as default assumptions. The assumptions are viewed as default assertions, any instance of which may be used as an axiom so long as it is consistent with the current set of observations. For example, under this interpretation rule (1) of fig. 2 asserts that "one who has a cold can be assumed to have sneezing, as long as that is consistent with the facts and observations." In other words, by treating unmarked formulas as defaults, the universal generalization normally entailed by a classical first order interpretation is not required. The formal foundation of this default interpretation is already exploited in Theorist proper [Poole86b], and is similar to the interpretation of defaults described in the Residue system of Finger and Genesereth [Finger85].

Both of these interpretations provide the rudiments of a procedure for preferring one diagnosis over another. For example, consider the possible diagnoses of the observations

$$has(O, fever) \wedge has(O, runny\text{-}nose) \wedge has(O, cold)$$

All of the necessary symptoms are observed for the known afflictions so that all of $has(O, influenza)$, $has(O, hayfever)$ and $has(O, cold)$ are possible diagnoses. If we confirm the absence of the sneezing, coughing, headache and diarrheoa we would intuitively prefer the hayfever diagnosis over the others on the basis of volume of evidence. In probabilistic terms, we want the negative affect of the confirmed absence of relevant symptoms to make the probability of runny-nose and watery-eyes to exceed that of the other two entailed symptoms. Similarly for the default interpretation, where the number of inconsistent default symptoms negatively impacts the "quality" of the hypothesized diagnoses of cold and influenza.

6. Conclusions

In general, automatic theory formation of the form suggested by the Theorist system requires that the representation language be able to express potential contradictions. The SLD proof procedure of Prolog is defined only for a language in which no contradiction is expressible, but a suitable interpretation of integrity constraints provides a simple notion of potential contradiction. Theorist-S is a very simple theory formation system that distinguishes from tacit symptoms to provide a way of denying theory membership to some possible hypotheses.

Diagnosis systems based on Theorist-S offer a potential advantage in that the relationship between diseases and symptoms can be expressed in a natural way independent of any qualifications that indicate how that information should be used. The simple annotation that provides a binary classification of symptoms can be exploited with a very simple proof mechanism based on SLD resolution. Furthermore, at least two possible interpretations of the classification both provide the necessary semantics for preferring one theory over another in an intuitively plausible way.

There are many problems that remain to be investigated, however. There are know difficulties with any system that uses a resolution proof procedure on clausal form, e.g., the problem of reverse Skolemization when attempting to establish consistency, and consistency verification requires a complete proof procedure whose inefficiencies have been explicitly avoided in the standard Prolog implementation of SLD resolution. Furthermore, it is not yet clear how to generalize either the probabilistic or default interpretation in a way that provides an effective and efficient method of preferring one diagnosis over another.

Despite these major difficulties, the simplicity of the theory formation procedure warrants further investigation as it so easily captures the intuition behind the diagnostic process.

Acknowledgements

Romas Aleliunas and Maarten van Emden suggested numerous improvements to an earlier draft of this paper. We are grateful to Ehud Shapiro and Jack Minker, for pointing us to some some relevant work that we had overlooked. This research was supported by National Sciences and Engineering Research Council of Canada grant A0894.

References

[Bowen85] K.A Bowen and T. Weinberg (1985), A meta-level extension of Prolog, *IEEE 1985 Symposium on Logic Programming*, July 15-18, Boston, Massachusetts, 48-53.

[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.

[Clark78] K.L. Clark (1978), Negation as failure, *Logic and Data Bases*, H. Gallaire and J. Minker (eds.), Plenum Press, New York, 293-322.

[Finger85] J.J. Finger and M.R. Genesereth (1985), Residue: a deductive approach to design synthesis, STAN-CS-85-1035, Computer Science Department, Stanford University, Stanford, California, January.

[Genesereth85] M.R. Genesereth (1985), The use of design descriptions in automated diagnosis, *Qualitative Reasoning about Physical Systems*, D.G. Bobrow (eds.), MIT Press, Cambridge, Massachusetts, 411-436.

[Goebel85a] R.G. Goebel (1985), DLOG: an experimental PROLOG-based database management system, *Proceedings of the IFIP Working Conference on Data Bases in the Humanities and Social Sciences*, R.F. Allen (ed.), Para-

digm Press, New York, 293-306.

[Goebel85b] R.G. Goebel (1985), A logic-based data model for the machine representation of knowledge, Ph.D. dissertation, Computer Science Department, The University of British Columbia, Vancouver, British Columbia, October, 253 pages.

[Hammond82] P. Hammond (1982), Logic programming for expert systems, DOC 82/4, Department of Computing, Imperial College of Science and Technology, University of London, March.

[Kahn84] K.M. Kahn and M. Carlsson (1984), The compilation of Prolog programs without the use of a Prolog compiler, *Proceedings of the International Conference on Fifth Generation Computer Systems*, November 6-9, Tokyo, Japan, 348-355.

[Kohli83] M. Kohli and J. Minker (1983), Intelligent control using integrity constraints, *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, August 22-26, University of Maryland/George Washington University, Washinton, D.C., 202-205.

[Lloyd84] J.W. Lloyd (1984), *Foundations of logic programming*, Springer-Verlag, New York.

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

[Meltzer73] B. Meltzer (1973), The programming of deduction and induction, *Artificial and Human Thinking*, A. Elithorn and D. Jones (eds.), Jossey-Bass, London, England, 19-33.

[Miyachi84] T. Miyachi, S. Kunifuji, H. Kitakami, K. Furukawa, A. Takeuchi, and H. Yokota (1984), A knowledge assimiliation method for logic databases, *Proceedings of the IEEE International Symposium on Logic Programming*, February 6-9, Atlantic City, New Jersey, 118-125.

[Nilsson84] N.J. Nilsson (1984), Probabilistic logic, Technical Note 321, Artificial Intelligence Center, SRI International, Menlo Park, California, February.

[Poole86a] D.L. Poole (1986), Default reasoning and diagnosis as theory formation, Technical Report CS-86-08, Department of Computer Science, University of Waterloo, Waterloo, Ontario, March.

[Poole86b] D.L. Poole, R.G. Goebel, and R. Aleliunas (1986), Theorist: a logical reasoning system for defaults and diagnosis, *Knowledge Representation*, N.J. Cercone and G. McCalla (eds.), Springer-Verlag, New York [invited chapter, submitted September 10, 1985].

[Pople77] H.E. Pople (1977), The formation of composite hypotheses in diagnostic problem solving: an exercise in synthetic reasoning, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, August 22-25, Cambridge, Massachusetts, 1030-1037.

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

[Shapiro83] E. Shapiro (1983), Logic programs with uncertainties: a tool for implementing rule-based systems, *Proceedings of IJCAI-83*, August 8-12,

Karlsruhe, Germany, 529-532.

[Shapiro81] E.Y. Shapiro (1981), An algorithm that infers theories from facts, *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August 24-28, The University of British Columbia, Vancouver, British Columbia, 446-451.

[Shapiro82] E. Y. Shapiro (1982), *Algorithmic program debugging*, MIT Press, Cambridge, Massachusetts.

[Sterling84] L. Sterling (1984), Logical levels of problem solving, *Proceedings of the Second International Logic Programming Conference*, July 2-6, Uppsala University, Uppsala, Sweden, 231-242.

[Takeuchi85] A. Takeuchi and Koichi Furukawa (1985), Partial evaluation of Prolog programs and its application to meta programming, Technical report TR-126, Institute for New Generation Computer Technology, Tokyo, Japan, September [to appear in *Lecture Notes in Computer Science*, Springer].

[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.