Logic Programming, Abduction and Probability: a top-down anytime algorithm for estimating prior and posterior probabilities

David Poole[*]
Department of Computer Science,
University of British Columbia,
Vancouver, B.C., Canada V6T 1Z2
poole@cs.ubc.ca

March 17, 1993

Abstract

Probabilistic Horn abduction is a simple framework to combine probabilistic and logical reasoning into a coherent practical framework. The numbers can be consistently interpreted probabilistically, and all of the rules can be interpreted logically. The relationship between probabilistic Horn abduction and logic programming is at two levels. At the first level probabilistic Horn abduction is an extension of pure Prolog, that is useful for diagnosis and other evidential reasoning tasks. At another level, current logic programming implementation techniques can be used to efficiently implement probabilistic Horn abduction. This forms the basis of an "anytime" algorithm for estimating arbitrary conditional probabilities. The focus of this paper is on the implementation.

[*]Scholar, Canadian Institute for Advanced Research

1 Introduction

Probabilistic Horn Abduction [22, 21, 23] is a framework for logic-based abduction that incorporates probabilities with assumptions. It is being used as a framework for diagnosis [22] that incorporates both pure Prolog and discrete Bayesian Networks [15] as special cases [21]. This paper is about the relationship of probabilistic Horn abduction to logic programming. This simple extension to logic programming provides a wealth of new applications in diagnosis, recognition and evidential reasoning [23].

This paper also presents a logic-programming solution to the problem in abduction of searching for the "best" diagnoses first. The main features of the approach are:

- We are using Horn clause abduction. The procedures are simple, both conceptually and computationally (for a certain class of problems). We develop a simple extension of SLD resolution to implement our framework.
- The search algorithms form "anytime" algorithms that can give an estimate of the conditional probability at any time. We do not generate the unlikely explanations unless we need to. We have a bound on the probability mass of the remaining explanations which allows us to know the error in our estimates.
- A theory of "partial explanations" is developed. These are partial proofs that can be stored in a priority queue until they need to be further expanded. We show how this is implemented in a Prolog interpreter in Appendix A.

2 Probabilistic Horn abduction

The formulation of abduction used is a simplified form of Theorist [25, 19] with probabilities associated with the hypotheses. It is simplified in being restricted to definite clauses with simple forms of integrity constraints (similar to that of Goebel et. al. [10]). This can also be seen as a generalisation of an ATMS [27] to be non-propositional.

The language is that of pure Prolog (i.e., definite clauses) with special disjoint declarations that specify a set of disjoint hypotheses with associated probabilities. There are some restrictions on the forms of the rules and the probabilistic dependence allowed. The language presented here is that of [23] rather than that of [22, 21].

The main design considerations were to make a language the simplest extension to pure Prolog that also included probabilities (not just numbers associated with rules, but numbers that follow the laws of probability, and so can be consistently interpreted as probabilities [23]). We are also assuming very strong independence assumptions; this is not intended to be a temporary restriction on the language that we want to eventually remove, but as a feature. We can represent any probabilistic information using only independent hypotheses [23]; if there is any dependence amongst hypotheses, we invent a new hypothesis to explain that dependency.

2.1 The language

Our language uses the Prolog conventions, and has the same definitions of variables, terms and atomic symbols.

Definition 2.1 A **definite clause** (or just "clause") is of the form: $a.$ or $a \leftarrow a_1 \wedge \cdots \wedge a_n.$ where a and each a_i are atomic symbols.

Definition 2.2 A **disjoint declaration** is of the form

$$disjoint([h_1 : p_1, \cdots, h_n : p_n]).$$

where the h_i are atoms, and the p_i are real numbers $0 < p_i \leq 1$ such that $p_1 + \cdots + p_n = 1$. Any variable appearing in one h_i must appear in all of the h_j (i.e., the h_i share the same variables). The h_i will be referred to as **hypotheses**.

Any ground instance of a hypothesis in one disjoint declaration, cannot be an instance of another hypothesis in any of the disjoint declarations (either the same declaration or a different declaration) nor can it be an instance of the head of any clause. This restricts the language so that we cannot encode arbitrary clauses as disjoint declarations. Each instance of each disjoint declaration is independent of all of the other instances of disjoint declarations and the clauses.

Definition 2.3 A **probabilistic Horn abduction theory** (which will be referred to as a "theory") is a collection of definite clauses and disjoint declarations.

Given theory T, we define

F_T the **facts**, is the set of definite clauses in T together with the clauses of the form

$$false \leftarrow h_i \wedge h_j$$

where h_i and h_j both appear in the same disjoint declaration in T, and $i \neq j$. Let F'_T be the set of ground instances of elements of F_T.

H_T to be the set of **hypotheses**, the set of h_i such that h_i appears in a disjoint declaration in T. Let H'_T be the set of ground instances of elements of H_T.

P_T is a function $H'_T \mapsto [0,1]$. $P_T(h'_i) = p_i$ where h'_i is a ground instance of hypothesis h_i, and $h_i : p_i$ is in a disjoint declaration in T.

Where T is understood from context, we omit the subscript.

Definition 2.4 [25, 18] If g is a closed formula, an **explanation** of g from $\langle F, H\rangle$ is a set D of elements of H' such that

- $F \cup D \models g$ and
- $F \cup D \not\models false$.

The first condition says that D is a sufficient cause for g, and the second says that D is possible.

Definition 2.5 A **minimal explanation** of g is an explanation of g such that no strict subset is an explanation of g.

2.2 Assumptions about the rule base

Probabilistic Horn abduction also contains some assumptions about the rule base. It can be argued that these assumptions are natural, and do not really restrict what can be represented [23]. Here we list these assumptions, and use them in order to show how the algorithms work.

The first assumption we make is about the relationship between hypotheses and rules:

Assumption 2.6 There are no rules with head unifying with a member of H.

Instead of having a rule implying a hypothesis, we can invent a new atom, make the hypothesis imply this atom, and all of the rules imply this atom, and use this atom instead of the hypothesis.

Assumption 2.7 (acyclicity) If F' is the set of ground instances of elements of F, then it is possible to assign a natural number to every ground atom such that for every rule in F' the atoms in the body of the rule are strictly less than the atom in the head.

This assumption is discussed by Apt and Bezem [1].

Assumption 2.8 The rules in F' for a ground non-assumable atom are covering.

That is, if the rules for a in F' are

$$\begin{array}{c} a \leftarrow B_1 \\ a \leftarrow B_2 \\ \vdots \\ a \leftarrow B_m \end{array}$$

if a is true, one of the B_i is true. Thus Clark's completion [4] is valid for every non-assumable. Often we get around this assumption by adding a rule

$$a \leftarrow some\_other\_reason\_for\_a$$

and making "$some\_other\_reason\_for\_a$" a hypothesis [23].

Assumption 2.9 The bodies of the rules in F' for an atom are mutually exclusive.

Given the above rules for a, this means that for each $i \neq j$, B_i and B_j cannot both be true in the domain under consideration[1]. We can make this true by adding extra conditions to the rules to make sure they are disjoint.

Associated with each possible hypothesis is a prior probability. We use this prior probability to compute arbitrary probabilities. Our final assumption is to assume that logical dependencies impose the only statistical dependencies on the hypotheses. In particular we assume:

Assumption 2.10 Ground instances of hypotheses that are not inconsistent (with F_T) are probabilistically independent.

The only ground instances of hypotheses that are inconsistent with F_T are those of the form $h_i\theta$ and $h_j\theta$, where h_i and h_j are different elements of the same disjoint declaration. Thus different disjoint declarations define independent hypotheses. Different instances of hypotheses are also independent. Note that the hypotheses in a minimal explanation are always logically independent. The language has been carefully set up so that the logic does not force any dependencies amongst the hypotheses. If we could prove that some hypotheses implied other hypotheses or their negations, the hypotheses could not be independent. The language is deliberately designed to be too weak to be able to state such logical dependencies between hypotheses.

See [23] for more justification of these assumptions.

2.3 Some Consequents of the Assumptions

Lemma 2.11 Under assumption 2.10, if there is at least ony hypothesis with a free variable, then distinct ground terms denote different individuals.

> **Proof:** If t_1 and t_2 are distinct ground terms, then $t_1 \neq t_2$. Otherwise, suppose $t_1 = t_2$. If $h(X)$ is a hypothesis, then $h(t_1)$ cannot be independent of $h(t_2)$ as they are logically equivalent, which violates assumption 2.10. $\Box$

[1]We do not insist that we are able to prove $neg(B_i \wedge b_j)$. Typically we cannot. This is a semantic restriction. There is, however, one syntactic check we can make. If there is a set of independent (see assumption 2.10) hypotheses from which both B_i and B_j can be derived then the assumption is violated.

The following lemma can be easily proved:

Lemma 2.12 Under assumptions 2.6 and 2.9, minimal explanations of ground atoms or conjunctions of ground atoms are mutually inconsistent.

Lemma 2.13 A minimal explanation of a ground atom cannot contain any free variables.

> **Proof:** If a minimal explanation of a ground atom contains free variables, there will be infinitely many ground explanations of the atom (substituting different ground terms for the free variables). Each of these ground explanations will have the same probability. Thus this probability must be zero (as we can sum the probabilities of disjoint formulae, and this sum must be less than or equal to one). This contradicts the fact that the explanations are minimal, and the hypotheses are independent, and each positive. □

Lemma 2.14 [5, 19] Under assumptions 2.6, 2.7 and 2.8, if $expl(g,T)$ is the set of minimal explanations of ground g from theory T, and $comp(T)$ is Clark's completion of the non assumables (and includes Clark's equational theory) then[2]

$$comp(T) \models \left(g \equiv \bigvee_{e_i \in expl(g,T)} e_i \right)$$

The following is a corollary of lemmata 2.14 and 2.12

Lemma 2.15 Under assumptions 2.6, 2.7, 2.8, 2.9 and 2.10, if $expl(g,T)$ is the set of minimal explanations of conjunction of atoms g from probabilistic Horn abduction theory T:

$$\begin{aligned} P(g) &= P\left(\bigvee_{e_i \in expl(g,T)} e_i \right) \\ &= \sum_{e_i \in expl(g,T)} P(e_i) \end{aligned}$$

[2]For the case we have here, Console et. al.'s results [5] can be extended to acyclic theories (the set of ground instances of the acyclic theory is a hierarchical theory, and we only need really consider the ground instances).

Thus to compute the prior probability of any g we sum the probabilities of the explanations of g. Poole [23] proves this directly using a semantics that incorporates the above assumptions.

To compute arbitrary conditional probabilities, we use the definition of conditional probability:

$$P(\alpha|\beta) = \frac{P(\alpha \wedge \beta)}{P(\beta)}$$

Thus to find arbitrary conditional probabilities $P(\alpha|\beta)$, we find $P(\beta)$, which is the sum of the explanations of β, and $P(\alpha \wedge \beta)$ which can be found by explaining α from the explanations of β. Thus arbitrary conditional probabilities can be computed from summing the prior probabilities of explanations.

It remains only to compute the prior probability of an explanation D of g.

Under assumption 2.10, if $\{h_1, \cdots, h_n\}$ are part of a minimal explanation, then

$$P(h_1 \wedge \cdots \wedge h_n) \quad = \quad \prod_{1=1}^{n} P(h_i)$$

To compute the prior of the minimal explanation we multiply the priors of the hypotheses. The posterior probability of the explanation is proportional to this.

2.4 An example

In this section we show an example that we use later in the paper. It is intended to be as simple as possible to show how the algorithm works.

Suppose we have the rules and hypotheses[3]:

```
rule((a :- b, h)).
rule((a :- q,e)).
rule((q :- h)).
rule((q :- b,e)).
rule((h :- b, f)).
```

[3]Here we have used a syntax `rule((H :- B))` to represent the rule $H \leftarrow B$. This is the syntax that is accepted by the code in Appendix A.

```
rule((h :- c, e)).
rule((h :- g, b)).
disjoint([b:0.3,c:0.7]).
disjoint([e:0.6,f:0.3,g:0.1]).
```

There are four minimal explanations of a, namely $\{c, e\}$, $\{b, e\}$, $\{f, b\}$ and $\{g, b\}$.

The priors of the explanations are as follows:

$$P(c \wedge e) = 0.7 \times 0.6 = 0.42.$$

Similarly $P(b \wedge e) = 0.18$, $P(f \wedge b) = 0.09$ and $P(g \wedge b) = 0.03$. Thus

$$P(a) = 0.42 + 0.18 + 0.09 + 0.03 = 0.72$$

There are two explanations of $e \wedge a$, namely $\{c, e\}$ and $\{b, e\}$. Thus $P(e \wedge a) = 0.60$. Thus the conditional probability of e given a is $P(e|a) = 0.6/0.72 = 0.833$.

What is important about this example is that all of the probabilistic calculations reduce to finding the probabilities of explanations.

2.5 Tasks

The following tasks are what we expect to implement:

1. Generate the explanations of some goal (conjunction of atoms), in order.

2. Estimate the prior probability of some goal. This is implemented by enumerating some of the explanations of the goal.

3. Estimate the posterior probabilities of the explanations of a goal (i.e., the probabilities of the explanations given the goal).

4. Estimate the conditional probability of one formula given another. That is, determining $P(\alpha|\beta)$ for any α and β.

Each of these will be computed using the preceding task. The last task is the essential task that we need in order to make decisions under uncertainty.

All of these will be implemented by enumerating the explanations of a goal, and estimating the probability mass in the explanations that have not been enumerated. It is this problem that we consider for the next few sections, and then return to the problem of the tasks we want to compute.

3 A top-down proof procedure

In this section we show how to carry out a best-first search of the explanations. In order to do this we build a notion of a partial proof that we can add to a priority queue, and restart when necessary.

3.1 SLD-BF resolution

In this section we outline an implementation based on logic programming technology and a branch and bound search.

The implementation keeps a priority queue of sets of hypotheses that could be extended into explanations ("partial explanations"). At any time the set of all the explanations is the set of already generated explanations, plus those explanations that can be generated from the partial explanations in the priority queue.

Definition 3.1 a **partial explanation** is a structure

$$\langle g \leftarrow C, D \rangle$$

where g is an atom (or conjunction of atoms), C is a conjunction of atoms and D is a set of hypotheses.

Figure 1 gives an algorithm for finding explanations of q in order of probability (most likely first).

We have the following data structures:

Q is a set of partial explanations (implemented as a priority queue).

Π is a set of explanations of g that have been generated (initially empty).

NG is the set of pairs $\langle h_i, h_j \rangle$ such that h_i and h_j are different hypotheses in a disjoint declaration (N.B. h_i and h_j share the same variables).

At each step we choose an element

$$\langle g \leftarrow C, D \rangle$$

of the priority queue Q. Which element is chosen depends on the search strategy (e.g., we could choose the element with maximum prior probability of D).

```
$Q := \{\langle g \leftarrow g, \{\}\rangle\}$;
$\Pi := \{\}$;
repeat
        choose and remove best $\langle g \leftarrow C, D\rangle$ from $Q$;
        if $C = \mathit{true}$
           then if $\mathit{good}(D)$ then $\Pi := \Pi \cup \{D\}$ endif
           else Let $C = a \wedge R$
                 for each $\mathit{rule}(h \leftarrow B)$ where $\mathit{mgu}(a,h) = \theta$
                             $Q := Q \cup \{\langle g \leftarrow B \wedge R, D\rangle\,\theta\}$ ;
                 if $a = h\theta$ where $h \in H$ and $\mathit{good}(\{a\} \cup D)$
                             then $Q := Q \cup \{\langle g \leftarrow R, \{a\} \cup D\rangle\}$
                 endif
        endif
until $Q = \{\}$
where $\mathit{good}(D) \equiv (\forall d_1, d_2 \in D\ \ \nexists \eta \in NG\ \exists \phi\ \ \langle d_1, d_2\rangle = \eta\phi)$
```

Figure 1: SLD-BF Resolution to find explanations of g in order.

We have an explanation when C is the empty conjunction (represented here as $true$). In this case D is added to the set Π of already generated explanations.

Otherwise, suppose C is conjunction $a \wedge R$.

There are two operations that can be carried out. The first is a form of SLD resolution [12], where for each rule

$$h \leftarrow b_1 \wedge \cdots \wedge b_n$$

in F, such that h and a have most general unifier θ, we generate the partial explanation

$$\langle g \leftarrow b_1 \wedge \cdots \wedge b_n \wedge R, D \rangle \, \theta$$

and add it to the priority queue.

The second operation is used when a is an instance (not necessarily ground) of a possible hypothesis. In this case we produce the partial explanation

$$\langle g \leftarrow R, \{a\} \cup D \rangle$$

and add it to Q. We only do this if $\{a\} \cup D$ is consistent. To check consistency, we use the set NG of pairs of hypotheses that appear in the same disjoint declaration (corresponding to *nogoods* in an ATMS [27]). Unlike in an ATMS, this set can be built at compile time from the disjoint declarations. A set of hypotheses is inconsistent only if there are two elements of the set that are instances of a nogood.

This procedure will enumerare the explanations (e.g., in order of likelihood). Its correctness is based on the meaning of a partial explanation

Definition 3.2 A partial explanation $\langle g \leftarrow C, D \rangle$ is **valid** with respect to $\langle F, H \rangle$ if

$$F \models D \wedge C \Rightarrow g$$

Lemma 3.3 Every partial explanation in the queue Q is valid with respect to $\langle F, H \rangle$.

> **Proof:** This is proven by induction on the number of times through the loop.
>
> It is trivially true initially as $q \Rightarrow q$ for any q.

There are two cases where elements are added to Q. In the first case (the "rule" case) we know

$$F \models D \wedge R \wedge a \Rightarrow g$$

by the inductive assumption, and so

$$F \models (D \wedge R \wedge a \Rightarrow g)\theta$$

We also know

$$F \models (B \Rightarrow h)\theta$$

As $a\theta = h\theta$, by a simple resolution step we have

$$F \models (D \wedge R \wedge B \Rightarrow g)\theta.$$

The other case is when $a \in H$. By the induction step

$$F \models D \wedge (a \wedge R) \Rightarrow g$$

and so

$$F \models (D \wedge a) \wedge R \Rightarrow g$$

If D only contains elements of H and a is an element of H then $\{a\} \cup D$ only contains elements of H. $\square$

It is now trivial to show the following:

Corollary 3.4 Every element of Π in Figure 1 is an explanation of q.

Although the correctness of the algorithm does not depend on which element of the queue we choose at any time, the efficiency may (particularly when we do not run the algorithm to completion; see Section 4.1). We could use a best-first strategy [14] where we choose the *best* partial explanation based on the following ordering of partial explanations: partial explanation $\langle g_1 \leftarrow C_1, D_1\rangle$ is better than $\langle g_2 \leftarrow C_2, D_2\rangle$ if $P(D_1) \geq P(D_2)$[4]. It is simple to show that "better than" is a partial ordering. When we choose a "best" partial explanation we choose a minimal element of the partial ordering;

[4]By D_i we mean the conjunction of the elements of D_i. The conjunction of the elements of the empty set is true, thus $P(\{\}) = 1$ which will always be a best partial explanation.

where there are a number of equally minimal partial explanations, we can choose any one. When we follow this definition of "best", we enumerate the minimal explanations of q in order of probability.

Other search strategies can be used, for example, depth-bounded search and iterative deepening search. The algorithm is independent of the search strategy used, but for concreteness of the examples and the discussion we assume the best-first strategy is used.

3.2 Our example

In this section we show how the simple example in Section 2.4 is handled by the best-first proof process.

The following is the sequence of values of Q each time through the loop (where there are a number of minimal explanations, we choose the element that was added last):

$$
\begin{array}{l}
\{\langle a \leftarrow a, \{\}\rangle\} \\
\{\langle a \leftarrow b \wedge h, \{\}\rangle, \langle a \leftarrow q \wedge e, \{\}\rangle\} \\
\{\langle a \leftarrow q \wedge e, \{\}\rangle, \langle a \leftarrow h, \{b\}\rangle\} \\
\{\langle a \leftarrow h \wedge e, \{\}\rangle, \langle a \leftarrow b \wedge e \wedge e, \{\}\rangle, \langle a \leftarrow h, \{b\}\rangle\} \\
\{\langle a \leftarrow b \wedge f \wedge e, \{\}\rangle, \langle a \leftarrow c \wedge e \wedge e, \{\}\rangle, \\
\quad \langle a \leftarrow g \wedge b \wedge e, \{\}\rangle, \langle a \leftarrow b \wedge e \wedge e, \{\}\rangle, \langle a \leftarrow h, \{b\}\rangle\} \\
\{\langle a \leftarrow c \wedge e \wedge e, \{\}\rangle, \langle a \leftarrow g \wedge b \wedge e, \{\}\rangle, \\
\quad \langle a \leftarrow b \wedge e \wedge e, \{\}\rangle, \langle a \leftarrow f \wedge e, \{b\}\rangle, \langle a \leftarrow h, \{b\}\rangle\} \\
\{\langle a \leftarrow g \wedge b \wedge e, \{\}\rangle, \langle a \leftarrow b \wedge e \wedge e, \{\}\rangle, \langle a \leftarrow e \wedge e, \{c\}\rangle, \\
\quad \langle a \leftarrow f \wedge e, \{b\}\rangle, \langle a \leftarrow h, \{b\}\rangle\} \\
\{\langle a \leftarrow b \wedge e \wedge e, \{\}\rangle, \langle a \leftarrow e \wedge e, \{c\}\rangle, \langle a \leftarrow f \wedge e, \{b\}\rangle, \\
\quad \langle a \leftarrow h, \{b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle\} \\
\{\langle a \leftarrow e \wedge e, \{c\}\rangle, \langle a \leftarrow e \wedge e, \{b\}\rangle, \langle a \leftarrow f \wedge e, \{b\}\rangle, \\
\quad \langle a \leftarrow h, \{b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle\} \\
\{\langle a \leftarrow e, \{e, c\}\rangle, \langle a \leftarrow e \wedge e, \{b\}\rangle, \langle a \leftarrow f \wedge e, \{b\}\rangle, \\
\quad \langle a \leftarrow h, \{b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle\} \\
\{\langle a \leftarrow true, \{e, c\}\rangle, \langle a \leftarrow e \wedge e, \{b\}\rangle, \langle a \leftarrow f \wedge e, \{b\}\rangle, \\
\quad \langle a \leftarrow h, \{b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle\}
\end{array}
$$

Thus the first, and most likely explanation is $\{e,c\}$.

$$\begin{array}{l}
\{\langle a \leftarrow e \wedge e, \{b\}\rangle, \langle a \leftarrow f \wedge e, \{b\}\rangle, \langle a \leftarrow h, \{b\}\rangle, \\
\quad \langle a \leftarrow b \wedge e, \{g\}\rangle\} \\
\langle a \leftarrow f \wedge e, \{b\}\rangle, \langle a \leftarrow h, \{b\}\rangle, \langle a \leftarrow e, \{e,b\}\rangle, \\
\quad \{\langle a \leftarrow b \wedge e, \{g\}\rangle\} \\
\{\langle a \leftarrow h, \{b\}\rangle, \langle a \leftarrow e, \{e,b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle, \\
\quad \langle a \leftarrow e, \{f,b\}\rangle\} \\
\{\langle a \leftarrow b \wedge f, \{b\}\rangle, \langle a \leftarrow c \wedge e, \{b\}\rangle, \langle a \leftarrow g \wedge b, \{b\}\rangle, \\
\quad \langle a \leftarrow e, \{e,b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle, \langle a \leftarrow e, \{f,b\}\rangle\} \\
\{\langle a \leftarrow f, \{b\}\rangle, \langle a \leftarrow c \wedge e, \{b\}\rangle, \langle a \leftarrow g \wedge b, \{b\}\rangle, \\
\quad \langle a \leftarrow e, \{e,b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle, \langle a \leftarrow e, \{f,b\}\rangle\} \\
\{\langle a \leftarrow c \wedge e, \{b\}\rangle, \langle a \leftarrow g \wedge b, \{b\}\rangle, \langle a \leftarrow e, \{e,b\}\rangle, \\
\quad \langle a \leftarrow b \wedge e, \{g\}\rangle, \langle a \leftarrow true, \{f,b\}\rangle, \langle a \leftarrow e, \{f,b\}\rangle\}
\end{array}$$

Here the algorithm effectively prunes the top partial explanation as $\langle c, b\rangle$ forms a nogood.

$$\begin{array}{l}
\{\langle a \leftarrow g \wedge b, \{b\}\rangle, \langle a \leftarrow e, \{e,b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle, \\
\quad \langle a \leftarrow true, \{f,b\}\rangle, \langle a \leftarrow e, \{f,b\}\rangle\} \\
\{\langle a \leftarrow e, \{e,b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle, \langle a \leftarrow true, \{f,b\}\rangle, \\
\quad \langle a \leftarrow e, \{f,b\}\rangle, \langle a \leftarrow b, \{g,b\}\rangle\}\} \\
\{\langle a \leftarrow true, \{e,b\}\rangle, \langle a \leftarrow b \wedge e, \{g\}\rangle, \langle a \leftarrow true, \{f,b\}\rangle, \\
\quad \langle a \leftarrow e, \{f,b\}\rangle, \langle a \leftarrow b, \{g,b\}\rangle\}
\end{array}$$

We have now found the second most likely explanation, namely $\{e,b\}$.

$$\begin{array}{l}
\{\langle a \leftarrow b \wedge e, \{g\}\rangle, \langle a \leftarrow true, \{f,b\}\rangle, \langle a \leftarrow e, \{f,b\}\rangle, \\
\quad \langle a \leftarrow b, \{g,b\}\rangle\} \\
\{\langle a \leftarrow true, \{f,b\}\rangle, \langle a \leftarrow e, \{f,b\}\rangle, \langle a \leftarrow e, \{g,b\}\rangle, \\
\quad \langle a \leftarrow b, \{g,b\}\rangle\}
\end{array}$$

We have thus found the third explanation $\{f,b\}$.

$$\begin{array}{l}
\{\langle a \leftarrow e, \{f,b\}\rangle, \langle a \leftarrow e, \{g,b\}\rangle, \langle a \leftarrow b, \{g,b\}\rangle\} \\
\{\langle a \leftarrow e, \{g,b\}\rangle, \langle a \leftarrow b, \{g,b\}\rangle\} \\
\{\langle a \leftarrow b, \{g,b\}\rangle\} \\
\{\langle a \leftarrow true, \{g,b\}\rangle\}
\end{array}$$

The fourth explanation is $\{g,b\}$. There are no more partial explanations and the process stops.

4 Discussion

4.1 Probabilities in the queue

The point of the algorithm is to not let it run to completion, but to form the basis of an "anytime" algorithm [2], which can at any stage give an estimate of prior and posterior probabilities, and also provide an error guarantee.

We would like to give an estimate for $P(g)$ after having generated only a few of the most likely explanations of g, and get some estimate of our error. We can use Π and Q to estimate the prior probabilities of g and to give a bound on the error of our estimate.

If $\langle g \leftarrow C, D\rangle$ is in the priority queue, then it can possibly be used to generate explanations $D_1, \cdots, D_n$. Each D_i will be of the form $D \cup D_i'$. We can place a bound on the probability mass of all of the D_i, by

$$\begin{aligned} P(D_1 \vee \cdots \vee D_n) &= P(D \wedge (D_1' \vee \cdots \vee D_n')) \\ &\leq P(D) \end{aligned}$$

Given this upper bound, we can determine an upper bound for $P(g)$, where $\{e_1, e_2, \ldots\}$ is the set of all minimal explanations of g:

$$\begin{aligned} P(g) &= P(e_1 \vee e_2 \vee \cdots) \\ &= P(e_1) + P(e_2) + \cdots \\ &= \left(\sum_{e_i \text{ found}} P(e_i) \right) + \left(\sum_{e_j \text{ to be generated}} P(e_j) \right) \end{aligned}$$

We can easily compute the first of these sums, and can put upper and lower bounds on the second. This means that we can put a bound on the range of probabilities of a goal based on finding just some of the explanations of the goal. Suppose we have goal g, and we have generated explanations Π. Let

$$P_\Pi = \sum_{D \in \Pi} P(D)$$

$$P_Q = \sum_{D : \langle g \leftarrow C, D\rangle \in Q} P(D)$$

where Q is the priority queue.

We then have

Theorem 4.1 At the start of any iteration of the repeat loop in Figure 1, the prior probability of g can be bounded as follows:

$$P_\Pi \leq P(g) \leq P_\Pi + P_Q$$

As the computation progresses, the probability mass in the queue P_Q approaches zero[5] and we get a better refinement on the value of $P(g)$. This thus forms the basis of an "anytime" algorithm for Bayesian networks.

4.2 Conditional Probabilities

We can also use the above procedure to compute conditional probabilities. Suppose we are trying to compute the conditional probability $P(\alpha|\beta)$. This can be computed from the definition:

$$P(\alpha|\beta) = \frac{P(\alpha \wedge \beta)}{P(\beta)}$$

We compute the conditional probabilities by enumerating the minimal explanations of $\alpha \wedge \beta$ and β. Note that the minimal explanations of $\alpha \wedge \beta$ are explanations (not necessarily minimal) of β. We can compute the explanations of $\alpha \wedge \beta$, by trying to explain α from the explanations of β.

Figure 2 shows a modification of the algorithm of Figure 1 to interleave the explanation finding[6]. We collect the explanations of β as we generate them, and then expand these explanations to be explanations of $\alpha \wedge \beta$.

The elements of Q have either α or β at the left hand side of the '$\leftarrow$'. If they have β, then we are searching for explanations of β; when we have found one (say D) we want to explain α from this explanation of β. To do this, we put α on the left hand side of the '$\leftarrow$', and try to explain α starting from the explanation found for β (i.e., we add $\langle \alpha \leftarrow \alpha, D \rangle$ to Q). When we

[5]Note that the estimate given above does not always decrease. It is possible that the error estimate increases. Poole [24] gives a bottom-up procedure where convergence can be guaranteed.

[6]This is simplified in that we have removed the pruning of nogoods, and the use of substitutions (to make the presentation simpler). It is not difficult to add the pruning code, but you need to be careful to make sure that you appropriately prune the explanations (e.g., by not pruning explanations for α just because there is a simpler explanation for β). Substitutions are as in Figure 1.

```
$Q := \{\langle \beta \leftarrow \beta, \{\}\rangle\}$;
$\Pi_\beta := \{\}$;
$\Pi_{\alpha\wedge\beta} := \{\}$;
repeat
      choose and remove best $\langle g \leftarrow C, D\rangle$ from $Q$;
      if $C = true$
        then if $g = \alpha$
               then $\Pi_{\alpha\wedge\beta} := \Pi_{\alpha\wedge\beta} \cup \{D\}$
               else $\Pi_\beta := \Pi_\beta \cup \{D\}$;
                    $Q := Q \cup \{\langle \alpha \leftarrow \alpha, D\rangle\}$
               endif
        else Let $C = a \wedge R$;
               for each $rule(a \leftarrow B)$
                    $Q := Q \cup \{\langle g \leftarrow B \wedge R, D\rangle\, \theta\}$ ;
               if $a \in H$ then $Q := Q \cup \{\langle g \leftarrow R, \{a\} \cup D\rangle\}$ endif
      endif
until $Q = \{\}$
```

Figure 2: Simplified SLD-BF Resolution to find $P(\alpha|\beta)$ in order.

have found such an explanation (i.e., we have chosen $\langle \alpha \leftarrow true, D' \rangle$ from Q), then the corresponding assumption set (D') is an explanation of $\alpha \wedge \beta$.

Similarly to the case for computing prior probabilities, we are going to use Q, Π_β and $\Pi_{\alpha\wedge\beta}$ at the start of some iteration of the repeat loop to give us an estimate of posterior probabilities.

Let P^β be the sum of the probabilities of the explanations of β enumerated. That is,

$$P^\beta = \sum_{\pi \in \Pi_\beta} P(\pi).$$

Let $P^{\alpha\wedge\beta}$ be the sum of the probabilities of the explanations of $\alpha \wedge \beta$ generated. That is,

$$P^\beta = \sum_{\pi \in \Pi_{\alpha\wedge\beta}} P(\pi).$$

As before, let P_Q be the sum of the probabilities of the partial descriptions of the queue.

$$P_Q = \sum_{D:\langle g \leftarrow C, D \rangle \in Q} P(D)$$

Theorem 4.2 At the beginning of any iteration of the repeat loop in Figure 2, if $P^\beta > 0$ the conditional probability is bounded as follows:

$$\frac{P^{\alpha\wedge\beta}}{P^\beta + P_Q} \leq P(\alpha|\beta) \leq \frac{P^{\alpha\wedge\beta} + P_Q}{P^\beta}$$

Proof: We know (Theorem 4.1) that $P(\alpha \wedge \beta) \leq P^{\alpha\wedge\beta} + P_Q$, and $P(\beta) \geq P^\beta$, thus we have

$$P(\alpha|\beta) = \frac{P(\alpha \wedge \beta)}{P(\beta)} \leq \frac{P^{\alpha\wedge\beta} + P_Q}{P^\beta}$$

Similarly, $P(\alpha \wedge \beta) \geq P^{\alpha\wedge\beta}$, and $P(\beta) \leq P^\beta + P_Q$, thus we have

$$P(\alpha|\beta) = \frac{P(\alpha \wedge \beta)}{P(\beta)} \geq \frac{P^{\alpha\wedge\beta}}{P^\beta + P_Q}$$

□

The lower bound is the case where all of the partial descriptions in the queue go towards worlds implying β, but none of these also lead to α. The upper bound is the case where all of the elements of the queue go towards implying α, from the explanations already generated for β.

4.3 Consistency and subsumption checking

One problem that needs to be considered is the problem of what happens when there are free variables in the hypotheses generated. When we generate the hypotheses, there may be some instances of the hypotheses that are inconsistent, and some that are consistent. We know that every instance is inconsistent if the subgoal is subsumed by a nogood. This can be determined by substituting constants for the variables in the the subgoal, and finding if a subset unifies with a nogood.

We cannot prune hypotheses because an instance is inconsistent. However, when computation progresses, we may substitute a value for a variable that makes the partial explanation inconsistent. This problem is similar to the problem of delaying negation-as-failure derivations [13], and of delaying consistency checking in Theorist [20]. We would like to notice such inconsistencies as soon as possible. In the algorithm of Figure 1 we check for inconsistency each time a partial explanation is taken off the queue. There are cases where we do not have to check this explicitly, for example when we have done a resolution step that did not assign a variable. There is a trade-off between checking consistency and allowing some inconsistent hypotheses on the queue[7]. This trade-off is beyond the scope of this paper.

Note that the assumptions used in building the system imply that there can be no free variables in any explanation of a ground goal (otherwise we have infinitely many disjoint explanations with bounded probability). Thus delaying subgoals eventually grounds all variables.

4.4 Iterative deepening

In many search techniques we often get much better space complexity and asymptotically the same time complexity by using an iterative deepening version of a search procedure [11]. An iterative deepening version of the best-first search procedure is exactly the same as the iterative deepening version of A* [11] with the heuristic function of zero. The algorithm of Figure 1 is given at a level of abstraction which does not preclude iterative deepening.

For our experimental implementations, we have used an interesting variant of iterative deepening. Our queue is only a “virtual queue” and we

[7]We have to check the consistency at some time. This could be as late as just before the explanation is added to Π.

only physically store partial explanations with probability greater than some threshold. We remember the mass of the whole queue, including the values we have chosen not to store. When the queue is empty, we decrease the threshold. We can estimate the threshold that we need for some given accuracy. This speeds up the computation and requires less space.

4.5 Recomputing subgoals

One of the problems with the above procedure is that it recomputes explanations for the same subgoal. If s is queried as a subgoal many times then we keep finding the same explanations for s. This has more to do with the notion of SLD resolution used than with the use of branch and bound search.

We are currently experimenting with a top-down procedure where we remember computation that we have computed, forming "lemmata". This is similar to the use of memo functions [30] or Earley deduction [16] in logic programming, but we have to be very careful with the interaction between making lemmata and the branch and bound search, particularly as there may be multiple answers to any query, and just because we ask a query does not mean we want to solve it (we may only want to bound the probability of the answer).

One of the reasons that we wanted to present this at a high level of abstraction rather than as an iterative deepening version (such as [31]) is to allow for such ideas as creating lemmata. Based on our experience, there are some cases where the lemma version is faster than even one iteration of the depth-bounded search needed for an iterative deepening solution.

There is some connection between the idea of saving recomputing subgoals and D'Ambroisio's [7] top-down search algorithm for computing posterior probabilities in Bayesian networks. He carries out a precompilation to determine the shared structure, and then does a top down search. His representation language, however is not as rich as the language presented here.

4.6 Bounding the priority queue

Another problem with the above procedure that is not solved by lemmatisation is that the bound on the priority queue can become quite large (i.e., greater than one). Some bottom-up procedures [24], can be engineered to have a tighter estimate of the probability mass of the queue (in particular a monotonically non-increasing estimate). See [24] for a description of a bottom-up procedure that can be compared to the top-down procedure in this paper. In [24] an average case analysis is given on the bottom-up procedure; while this is not an accurate estimate for the top-down procedure, the case where the bottom-up procedure is efficient [24] is the same case where the top-down procedure works well; that is where there are normality conditions that dominate the probability of each hypothesis (i.e., where all of the probabilities are near one or near zero).

The bottom up procedures work only for less general propositional languages. The efficiency is analogous to the difference between forward and backward chaining on propositional Horn clauses. Bottom-up, we can compute the set of consequences of a set of propositional Horn clauses in time linear in the number of clauses (we only need to chain on each clause once). Backward search, however, can take exponential time for a particular goal. Even given this, backward chaining approaches are still the most popular (e.g., the use of Prolog). It is for the same reasons that the top-down approach of this paper is useful.

5 Comparison with other systems

There are many other proposals for logic-based abduction schemes (e.g., [26, 6, 10, 18]). These, however, consider that we either find an arbitrary explanation or find all explanations. In practice there are prohibitively many of these. It is also not clear what to do with all of the explanations; there are too many to give to a user, and the costs of determining which of the explanations is the "real" explanation (by doing tests [28]) is usually not outweighed by the advantages of finding the real explanation. This is why it is important to take into account probabilities. We then have a principled reason for ignoring many explanations. Probabilities are also the right tool to use when we really are unsure as to whether something is true or not. For evidential reasoning tasks (e.g., diagnosis and recognition) it is not up to us to decide whether some hypothesis is true or not; all we have is probabilities and evidence to work out what is most likely true. Similar considerations motivated the addition of probabilities to consistency-based diagnosis [8].

There are many algorithms for abduction in logic programming (e.g., [9, 25, 31, 29]), that are similar to the resolution steps of Section 3.1. The main advances of this paper are notion of an anytime algorithm, the explicit use of the priority queue, the estimation of the prior and posterior probabilities (including error bounds), and the use of two explanations being found at once for conditional probabilities (Section 4.2).

Perhaps the closest work to that presented here is that of Stickel [31]. His is an iterative deepening search for the lowest cost explanation, and at that level can be seen as an iterative-deepening version of the algorithm of Figure 1. The problem he is trying to solve is to find the least cost explanation. He does not consider probabilities (but Charniak and Shimony [3] have interpreted cost-based abduction in terms of $-\log$ probabilities), nor does his algorithm form an anytime algorithm that can estimate probability bounds, with a given error.

6 Using existing logic programming technology

In this section we show how the branch and bound search can be implemented in Prolog. The basic idea is that when we are choosing a partial explanation to explore, we can choose any of those with maximum probability. If we choose the last one when there is more than one, we carry out a depth-first search much like normal Prolog, except when making assumptions. We only add to the priority queue when making assumptions, and let Prolog do the searching when we are not.

6.1 Remaining subgoals

Consider what subgoals remain to be solved when we are trying to solve a goal. Consider the clause:

$$h \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_m.$$

Suppose R is the conjunction of subgoals that remain to be solved after h in the proof. If we are using the leftmost reduction of subgoals, then the conjunction of subgoals remaining to be solved after subgoal b_i is

$$b_{i+1} \wedge \cdots \wedge b_m \wedge R$$

The total information of the proof is contained in the partial explanation at the point we are in the proof, i.e., in the remaining subgoals, current hypotheses and the associated answer. The idea we exploit is to make this set of subgoals explicit by adding an extra argument to each atomic symbol that contains all of the remaining subgoals.

6.2 Saving partial proofs

We would like for there to be enough information within each subgoal to prove the top level goal it was created to solve. When we have a hypothesis that needs to be assumed, the remaining subgoals and the current hypotheses form a partial explanation which we save on the queue. We then fail the current subgoal and look for another solution. If there are no solutions found (i.e., the top level computation fails), we can choose a saved subgoal (according to the order given in section 3.1), and continue the search.

Suppose in our proof we select a possible hypothesis h of cost $P(\{h\})$ with U being the conjunction of goals remaining to be solved, and T the set of currently assumed hypotheses with cost $P(T)$. We only want to consider this as a possible contender for the best solution if $P(\{h\} \cup T)$ is the minimal cost of all proofs being considered. The minimal cost proofs will be other proofs of cost $P(T)$. These can be found by failing the current subgoal. Before we do this we need to add U, with hypotheses $\{h\} \cup T$ to the priority queue. When the proof fails we know there is no proof with the current set of hypotheses; we remove the partial proof with minimal cost from the priority queue, and continue this proof.

We do a branch and bound search over the partial explanations, but when the priorities are equal, we use Prolog's search to prefer the last added. The overhead on the resolution steps is low; we only have to do a couple more simple unifications (a free variable with a term). The main overhead occurs when we reach a hypothesis. Here we store the hypotheses and remaining goals on a priority queue and continue or search by failing the current goal. This is quick (if we implement the priority queue efficiently); the overhead needed to find all proofs is minimal.

Appendix A gives code necessary to run the search procedure.

7 Conclusion

This paper has considered a logic programming approach that uses a mix between depth-first and branch-and-bound search strategies for abduction where we want to consider probabilities, and only want to generate the most likely explanations. The underlying language [23] is a superset of pure Prolog, and also an extension of Bayesian networks [15]. We thus have a logic programming solution to the problem of inference (finding posterior probabilities) in Bayesian networks, that turns out to be very different to the sorts of algorithms used for Bayesian networks [15]. This complements the work on other search algorithms in Bayesian networks [24], which can be seen as bottom-up versions of the top-down search algorithm presented here.

A Prolog interpreter

This appendix gives a brief overview of a meta-interpreter. Hopefully it is enough to be able to build a system. Our implementation contains more bells and whistles, but the core of it is here.

N.B. This interpreter does not do an occurs check. Thus it is not sound. We consider this is an important problem, but the solution [17] is orthogonal to the issues in this paper, and will complicate the presentation.

A.1 Prove

$$prove(G, T_0, T_1, C_0, C_1, U)$$

means that G can be proven with current assumptions T_0, resulting in assumptions T_1, where C_i is the probability of T_i, and U is the set of remaining subgoals.

The first rule defining *prove* is a special purpose rule for the case where we have found an explanation; this reports on the answer found.

```
prove(ans(A),T,T,C,C,_) :- !,
   ans(A,T,C).
```

The remaining rules are the real definition, that follow a normal pattern of Prolog meta-interpreters [30].

```
prove(true,T,T,C,C,_) :- !.
prove((A,B),T0,T2,C0,C2,U) :- !,
   prove(A,T0,T1,C0,C1,(B,U)),
   prove(B,T1,T2,C1,C2,U).
prove(H,T,T,C,C,_) :-
   hypothesis(H,PH),
   member(H,T),!.
prove(H,T,[H|T],C,C1,U) :-
   hypothesis(H,PH),
   \+ (( member(H1,T), makeground((H,H1)),
         nogood(H,H1) )),
   C1 is C*PH,
   add_to_PQ(process([H|T],C1,U)),
   fail.
prove(G,T0,T1,C0,C1,U) :-
   rul(G,B),
   prove(B,T0,T1,C0,C1,U).
```

A.2 Rule and disjoint declarations

We specify the rules of our theory using the declaration $rule(R)$ where R is the form of a Prolog rule. This asserts the rule produced.

```
rule((H :- B)) :- !,
   assert(rul(H,B)).
rule(H) :-
   assert(rul(H,true)).
```

The disjoint declaration forms *nogoods* and declares probabilities of hypotheses.

```
:- op( 500, xfx, : ).
disjoint([]).
disjoint([H:P|R]) :-
   assert(hypothesis(H,P)),
   make_disjoint(H,R),
   disjoint(R).
```

```
make_disjoint(_,[]).
make_disjoint(H,[H2 : _ | R]) :-
   assert(nogood(H,H2)),
   assert(nogood(H2,H)),
   make_disjoint(H,R).
```

A.3 Explaining

To find an explanation for a subgoal G we execute $explain(G)$. This creates a list of solved explanations and the probability mass found (in "done"), and creates an empty priority queue.

```
explain(G) :-
   assert(done([],0)),
   initQ,
   ex((G,ans(G)),[],1),!.
```

$ex(G, D, C)$ tries to prove G with assumptions D such that probability of D is C. If G cannot be proven, a partial proof is taken from the priority queue and restarted. This means that $ex(G, D, C)$ succeeds if there is some proof that succeeds.

```
ex(G,D,C) :-
   prove(G,D,_,C,_,true).
ex(_,_,_) :-
    remove_from_PQ(process(D,C,U)),!,
    ex(U,D,C).
```

We can report the explanations found, the estimates of the prior probability of the hypothesis, etc, by defining $ans(G, D, C)$, which means that we have found an explanation D of G with probability C.

```
ans(G,[],_) :-
   writeln([G,' is a theorem.']),!.
ans(G,D,C) :-
   allgood(D),
   qmass(QM),
   retract(done(Done,DC)),
   DC1 is DC+C,
    assert(done([expl(G,D,C)|Done],DC1)),
    TC is DC1 + QM,
    writeln(['Probability of ',G,
             ' = [',DC1,',',TC,']']),
    Pr1 is C / TC,
    Pr2 is C / DC1,
    writeln(['Explanation: ',D]),
    writeln(['Prior = ',C]),
    writeln(['Posterior = [',Pr1,',',Pr2,']']).
```

more is a way to ask for more answers. It will take the top priority partial proof and continue with it.

```
more :- ex(fail,_,_).
```

A.4 Auxiliary relations used

The following relations were also used. They can be divided into those for managing the priority queue, and those for managing the nogoods.

We assume that there is a global priority queue into which one can put formulae with an associated cost and from which one can extract the least cost formulae. We assume that the priority queue persists over failure of subgoals. It can thus be implemented by asserting into a Prolog database, but cannot be implemented by carrying it around as an extra argument in a meta-interpreter [30], for example. We would like both insertion and removal from the priority queue to be carried out in $\log n$ time where n is the number of elements of the priority queue. Thus we cannot implement it by having the queue asserted into a Prolog database if the asserting and retracting takes time proportional to the size of the objects asserted or retracted (which it seems to in the implementations we have experimented with).

Four operations are defined:

$$initQ$$

initialises the queue to be the empty queue, with zero queue mass.

$$add\_to\_PQ(process(D,C,U))$$

adds assumption set D, with probability C and remaining subgoals U to the priority queue. Adds C to the queue mass.

$$remove\_from\_PQ(process(D, C, U))$$

if the priority queue is not empty, extracts the element with highest probability (highest value of C) from the priority queue and reduces the queue mass by C. $remove\_from\_PQ$ fails if the priority queue is empty.

$$qmass(M)$$

returns the sum of the probabilities of elements of the queue.

We assume the relation for handling *nogoods*:

$$allgood(L)$$

fails if L has a subset that has been declared nogood.

Acknowledgements

Thanks to Andrew Csinger, Keiji Kanazawa and Michael Horsch for valuable comments on this paper. This research was supported under NSERC grant OGPOO44121, and under Project B5 of the Institute for Robotics and Intelligent Systems.

References

[1] K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3-4):335–363, 1991.

[2] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proc. 11th International Joint Conf. on Artificial Intelligence*, pages 979–984, Detroit, MI, August 1989.

[3] E. Charniak and S. E. Shimony. Probabilistic semantics for cost based abduction. In *Proc. 8th National Conference on Artificial Intelligence*, pages 106–111, Boston, July 1990.

[4] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[5] L. Console, D. Theseider Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[6] P. T. Cox and T. Pietrzykowski. General diagnosis by abductive inference. Technical Report CS8701, Computer Science, Technical University of Nove Scotia, Halifax, April 1987.

[7] B. D'Ambrosio. Real-time value-driven diagnosis. In *Proc. Third International Workshop on the Principles of Diagnosis*, pages 86–95, Rosario, Washington, October 1992.

[8] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proc. 11th International Joint Conf. on Artificial Intelligence*, pages 1324–1330, Detroit, August 1989.

[9] J. J. Finger and M. R. Genesereth. Residue: A deductive approach to design synthesis. Technical Report STAN-CS-85-1035, Department of Computer Science, Stanford University, Stanford, Cal., 1985.

[10] R. Goebel, K. Furukawa, and D. Poole. Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning. In E. Shapiro, editor, *Proc. Third International Conference on Logic Programming*, pages 211–222, London, July 1986.

[11] K. E. Korf. Depth-first iterative deepening: an optimal admissable tree search. *Artificial Intelligence*, 27(1):97–109, September 1985.

[12] J. W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation Series. Springer-Verlag, Berlin, second edition, 1987.

[13] L. Naish. *Negation and Control in Prolog.* Lecture Notes in Computer Science 238. Springer Verlag, 1986.

[14] J. Pearl. *Heuristics.* Addison-Wesley, Reading, MA, 1984.

[15] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, 1988.

[16] F. C. N. Pereira and S. M. Shieber. *Prolog and Natural-Language Analysis.* Center for the Study of Language and Information, 1987.

[17] D. A. Plaisted. The occur-check problem in Prolog. *New Generation Computing*, 2:309–322, 1984.

[18] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[19] D. Poole. Representing knowledge for logic-based diagnosis. In *International Conference on Fifth Generation Computing Systems*, pages 1282–1290, Tokyo, Japan, November 1988.

[20] D. Poole. Compiling a default reasoning system into Prolog. *New Generation Computing Journal*, 9(1):3–38, 1991.

[21] D. Poole. Representing Bayesian networks within probabilistic Horn abduction. In *Proc. Seventh Conf. on Uncertainty in Artificial Intelligence*, pages 271–278, Los Angeles, July 1991.

[22] D. Poole. Representing diagnostic knowledge for probabilistic Horn abduction. In *Proc. 12th International Joint Conf. on Artificial Intelligence*, pages 1129–1135, Sydney, August 1991.

[23] D. Poole. Probabilistic Horn abduction and Bayesian networks. Technical Report 92-20, Department of Computer Science, University of British Columbia, August 1992. To appear, *Artificial Intelligence* 1993.

[24] D. Poole. Search for computing posterior probabilities in Bayesian networks. Technical Report 92-24, Department of Computer Science, University of British Columbia, September 1992.

[25] D. Poole, R. Goebel, and R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer-Verlag, New York, NY, 1987.

[26] H. E. Pople, Jr. On the mechanization of abductive logic. In *Proc. 3rd International Joint Conf. on Artificial Intelligence*, pages 147–152, Stanford, August 1973.

[27] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems: preliminary report. In *Proc. 6th National Conference on Artificial Intelligence*, pages 183–188, Seattle, July 1987.

[28] A. Sattar and R. Goebel. Using crucial literals to select better theories. *Computational Intelligence*, 7(1):11–22, February 1991.

[29] M. Shanahan. Prediction is deduction, but explanation is abduction. In *Proc. 11th International Joint Conf. on Artificial Intelligence*, pages 1055–1060, Detroit, Mich., August 1989.

[30] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, Cambridge, MA, 1986.

[31] M. E. Stickel. A Prolog-like inference system for computing minimum-cost abductive explanations in natural language interpretations. Technical Note 451, SRI International, Menlo Park, CA, September 1988.