Experiments in the Theorist Paradigm
A collection of student papers on the Theorist Project

Edited by David Poole
Winter 1987

Research Report CS-87-30
May 1987

Experiments in the Theorist Paradigm
A collection of student papers on the Theorist Project

Edited by David Poole

Winter 1987

Introduction

The papers in this report are selected from final papers from the winter 1987 version the course cs786. They were all on aspects of reasoning in the Theorist framework. These papers take the range from proposals for extensions to the Theorist framework to implementation issues and applications.

These papers are based on the idea that much reasoning can be characterised as theory formation, and that theory formation from a fixed set of possible hypotheses is a useful tool.

There are papers which cover extensions to the paradigm, for example particular theory comparitors for preferring to use the most specific knowledge in inheritance systems (Kirby, Bryant, Lemieux), for solving the temporal prediction problem (Goodwin, Trudel), for solving the problem of preferring the most likely diagnosis (Neufeld), and arguing that the multiple extension problem is not a problem at all (Gagné). There are also papers on the implementation of Theorist: adding loop checking (Strooper), dependency-directed backtracking (Spencer), adding fixed and variable predicates (Young), and examining how default reasoning can be implemented using connectionism (Parsons). There are also papers on applications of the Theorist framework: using Theorist for user modelling (Van Arragon), microcode synthesis (Mahmood), and pronoun resolution (Song).

Each of these papers has gone through a refereeing process where everyone submitted a paper, reviewed three other papers, and then updated their papers in light of the reviewers comments. The authors themselves are each responsible for the correctness of the final version of their paper enclosed here.

TABLE OF CONTENTS

13. Retrieval and Microcode Synthesis of Register- Transfer Modules as Theory Formation
by Mossaddeq Mahmood

14. A Pronoun Resolution System Based on Theorist Framework
by Fei Song

Preferring the Most Specific Extension

Bruce Kirby
Centre for Philosophical Engineering and Trendy Formalisms
Logic Programming and Artificial Intelligence Group
Dept. of Computer Science
University of Waterloo

April 23, 1987

Abstract

One of the major problems of default reasoning is handling multiple extensions. While it is often possible to explain two contradictory facts, there are many cases where one or the other is preferred. In both temporal reasoning and inheritance hierarchies, a considerable amount of work has been done to differentiate between conflicting theories.

This paper presents some logical properties that any theory preference system should have. It analyses the theory preference semantics of Poole[Poo85] and Loui[Lou86] in terms of these criteria, and then proposes a semantics for choosing most specific extensions.

1 Introduction

Commonsense reasoning about the "real" world produces a number of problems. One of these is how to deal with non-monotonicity. First-order predicate calculus has the property of being strictly monotonic (i.e. If A $\models$ g, then A $\cup$ B $\models$ g). This does not allow conclusions to be drawn that are based on incomplete information, because additional information could negate them.

Poole[PGA85] gives a model theoretic definition of default reasoning, based on scientific theory formation. It attempts to explain a set of observations by developing a consistent theory from a fixed set of possible hypotheses which implies the observations. This formalisation, like others proposed[Rei80], can produce ambiguous answers. In many domains, there is an intuitively preferred answer and a number of formalisms have been developed to distinguish between conflicting theories.

In every domain for which theory preference mechanisms have been developed, the solutions were been developed in response to particular problem examples that have been presented. The semantics of a formalism cannot be proven correct; it can only be justified by our intuition and shown to work on examples. However, there are a number of properties that any theory preference system should have. If a theory comparator can be shown to have these properties, then it is more likely to be useful in the general case.

We present a set of criteria that any theory preference system must have to be useful as a reasoning system, and analyse inheritance hierarchies in terms of these criteria. Finally, we present propose a semantics for choosing among default extensions.

2 Theory Formation

2.1 Syntax

The syntax of the system is based on first order predicate calculus. Following the Prolog convention, variables will be written in upper case. Within facts, all free variables are considered to be universally quantified. A well formed formula (wff) is any conjunction, disjunction or negation of atomic symbols (they will usually be expressed in clausal form). A set of wffs is equated with its conjunction (i.e. S $\Leftrightarrow \bigwedge$ S). There are two inputs to the system: facts and defaults. Facts are specified as:

fact <wff>

These represent statements that are considered to be universally true within the world being considered. Defaults are of the form:

default *d* or

default *d*: <wff>

A default is a statement of typicality. It means that any ground instance of d can be assumed true, if it is consistent. The second form is equivalent to:

default d and
fact <wff> $\leftarrow d$

2.2 Semantics

The idea behind theory formation is: given a set of observations, find a consistent set of ground instances of the defaults that, along with the facts, implies the observations. Formally:

> If F is the set of wffs defined to be facts and Δ is the set of defaults, then D, a set of ground instances of elements of Δ, is said to be a *theory* that *explains* a ground wff w, if
> $F \cup D \models w$ and
> $F \cup D$ is consistent.

3 Multiple Extension Problem

Poole[Poo86] has shown how theory formation relates to Reiter's default logic. Unfortunately, it encounters the same problems with multiple extensions that default logic does. In some cases, it is possible to explain both p and $\neg p$. For example:

fact $bird(X) \leftarrow emu(X)$
default $bf(X) : flies(X) \leftarrow bird(X)$
default $enf(X) : \neg flies(X) \leftarrow emu(X)$
fact $emu(fred)$

{bf(fred)} is a theory that explains flies(fred), while {enf(fred)} is a theory that explains ¬flies(fred).

Reiter and Criscuolo[RC81], Poole[Poo85] and Loui[Lou86] all agree that the desired conclusion is ¬flies(fred). We prefer this answer because

the default used to prove it is more specific than the one used to prove flies(fred). Emus are a subclass of birds; so, they represent an exception to the general rule that birds fly.

4 Choosing the Most Specific Theory

The example above is an inheritance hierarchy with exceptions [ER83]. The class of emus is a subclass of the class of birds. In these cases, if there are contradictory conclusions possible, we want to use the default based on the smallest subclass. In other words, we want to use the default that requires the most specific information.

The problem is to develop a semantic characterisation of what makes one theory more specific than another. This would avoid the problems that arise from syntactic representations[Tou84]. We want to present a clear, intuitive definition of more specific: one that can be easily justified, and is implementable.

The intuitive concept behind theory preference based on specificity is: one theory is more specific than another, if the second theory is applicable whenever the first one is. Poole[Poo85] tried to formalise this concept by dividing the facts into two classes:

F_n those facts necessarily true in any world in the domain and

F_c those facts that are true in the particular world being considered. In the example given above, "bird(X) $\leftarrow$emu(X)" is a necessary fact, while "emu(fred)" is a contingent fact. We want to say that flies(fred) is more general than $\neg$flies(fred) because it would still be true if fred were only known to be a bird. That is, there is a fact F_p that implies flies(fred), but not $\neg$flies(fred). Formally:

Definition 1 *g is* predicted, *if there exists D, a set of instances of elements of Δ, that explains g, such that for every D' that explains $\neg g$, $\langle D, g\rangle \leq \langle D', \neg g\rangle$ and $\langle D', \neg g\rangle \not\leq \langle D, g\rangle$.*

Definition 2 *$\langle D, g\rangle \leq \langle D', \neg g\rangle$, if there exists a wff F_p such that:*
$F_c \cup D' \cup F_n \models F_p$ *and*
$F_p \cup D' \cup F_n \models \neg g$ *and*

$F_p \cup D \cup F_n \not\models g$ *and*
$F_p \cup D \cup F_n \not\models \neg g$

If we consider the bird/emu example again, we try to compare $\langle \{bf(fred)\}, flies(fred) \rangle$ against $\langle \{enf(fred)\}, \neg flies(fred) \rangle$. The choice F_p = bird(fred) shows that $\langle \{enf(fred)\}, \neg flies(fred) \rangle \leq \langle \{bf(fred)\}, flies(fred) \rangle$. However, there is no F_p that allows us to conclude ¬flies(fred), but not flies(fred), so the conclusion ¬flies(fred) is strictly more specific and is predicted.

For most simple inheritance problems, this theory comparator appears to produce the correct, most specific answer. However, when conjunctions of defaults are involved, it can produce unintuitive results.

Example 1

default $d_1 : g_1 \leftarrow a$;
default $d_2 : g_2 \leftarrow b$;
default $d_3 : \neg g_1 \leftarrow a$;
$F_c = a \wedge b$

If $F_p = \{a\}$ *then* $\langle \{d_1, d_2\}, g_1 \wedge g2 \rangle \leq \langle \{d_3\}, \neg(g_1 \wedge g2) \rangle$. *There is no* F_p *such that* $\langle \{d_3\}, \neg(g_1 \wedge g2) \rangle \leq \langle \{d_1, d_2\}, g_1 \wedge g2 \rangle$. *Therefore, according to our definition,* $g_1 \wedge g2$ *is predicted.*

The problem with this conclusion is that we predict $g_1 \wedge g_2$, but cannot predict g_1. While the default which explains $\neg(g_1 \wedge g_2)$ is applicable whenever the ones to explain $g_1 \wedge g_2$ are, the extra knowledge used to explain $g_1 \wedge g2$ is irrelevant to the proof of $\neg(g_1 \wedge g2)$. We do not want to allow extraneous information to affect whether we predict g_1. Another example, where we are unable to draw a conclusion when we should is:

Example 2

default $d_1 : b \leftarrow a$
default $d_2 : c \leftarrow b$
default $d_3 : \neg c \leftarrow a$
default $d_4 : e \leftarrow d$
default $d_5 : f \leftarrow e$
default $d_6 : \neg f \leftarrow d$
fact $g \leftarrow \neg c \wedge \neg f$

fact $\neg g \leftarrow c \wedge f$

$F_c = \{a \wedge d\}$

Clearly, $\neg c$ *and* $\neg f$ *are predicted, and* $F_p = \{a \wedge f\}$ *allows us to conclude the theory for g is more specific than that for* $\neg g$*. However, if we consider* $F_p = \{a \wedge \neg f\}$*, we can conclude that* $\neg g$ *is more specific than g. Despite the fact that we can predict* $\neg c$ *and* $\neg f$*, we are unable to predict g.*

5 When is Theory Preference Logical?

Individual instances of theory comparison cannot be considered entirely in isolation. Any theory preference system must be consistent with a number of logical criteria. We present some of these, and justify their need independent of any particular domain or theory comparator.

$D \leq D'$ means that D is preferred over D'. In inheritance hierarchies, this could mean that D is strictly more specific[Poo85]. In temporal reasoning, D would be strictly more chronologically persistent[GG86].

Definition 3 *Given F, a set of facts, and* Δ*, a set of defaults, then* $(F,\Delta) \vdash g$ *(g is predicted), if there exists a theory D that explains g, and for every theory* D' *that explains* $\neg g$*,* $D \leq D'$*.*

5.1 Consistency

Intuitively, $\vdash$ should be closely related to logical implication. At the very least, it should follow the following rules:

1. $(F,\Delta) \vdash g_1 \wedge g_2 \Leftrightarrow (F,\Delta) \vdash g_1$ and $(F,\Delta) \vdash g_2$
2. $(F,\Delta) \vdash g_1$ or $(F,\Delta) \vdash g_2 \Rightarrow (F,\Delta) \vdash g_1 \vee g_2$
3. $(F,\Delta) \vdash g \Rightarrow (F,\Delta) \nvdash \neg g$
4. $(F,\Delta) \vdash g_1$ and $(g_1 \supset g) \in F \Rightarrow (F,\Delta) \vdash g$

These rules ensure that the set of predicates predicted by the theory are a model of the facts. For a theory preference system to have any logical justification, it must, at the very least, be consistent with the facts known about the world. As the examples above showed, Poole's semantics for more specific does not have these properties.

5.2 Preferred subgoals

If g is predicted, then, clearly, we want to predict all logical consequences of g. However, the reverse is also true. We do not want to predict g, unless we can also predict the subgoals used to prove g. This is a similar concept to Loui's preferred premise defeater rule[Lou86]. He says, "If one conclusion is preferred to another, then other things being equal, conclusions based on the first should be preferred to conclusions based on the second." For example:

$(F,\Delta) \vdash g_1$
default $d_1 : g \leftarrow g_1$
default $d_2 : \neg g \leftarrow \neg g_1$

Even if it is possible to explain $\neg g_1$, we want to predict g, because it has the preferred subgoal.

Loui's rule, however, is not strong enough. If we add the following default to the emu/bird example:

default maybeflying(X): flying(X) $\leftarrow$flies(X)

then Poole's semantics will conclude flying(fred) because it cannot explain ¬flying(fred). However, the preferred extension is the one in which neither flying(fred) nor ¬flying(fred) is true.

6 Most Specific Theory Revisited

Poole's semantics for more specific do not satisfy the requirements listed above. As example 1 showed, it is possible to predict $g_1 \wedge g_2$, without predicting g_1. Example 2 violates rule (4) above. $(F,\Delta) \vdash \neg c \wedge \neg f$, and $(\neg c \wedge \neg f \supset g)$ is a fact, but $(F,\Delta) \nvdash g$.

Loui[Lou86] has developed a method of preferring the most specific theory based upon defeasible rules of inference. He has identified four "defeater" rules, each of which is designed to deal with one possible case that would make one goal preferable to another. Loui's defeater rules are capable of handling example 2 properly but, for example 1, his system produces the same answer as Poole's.

In order to deal with example 2, Loui suggested that Poole could iterate his theory preference system, performing the test at every step of the proof. While this is a solution to the examples above, it destroys the semantic

independence of the theory preference system. The goals predicted should depend on the facts and defaults, not on the proof procedure used to justify them.

What is needed is a formal semantics for preferring the most specific theory that, independent of the implementation, satisfies the criteria listed above. Our intuition of preferring the most specific theory is a simple one. There should be an equally simple semantics.

7 Preferring Extensions

7.1 Why are extensions useful?

Reiter's default logic generated an extension of the facts and defaults rather than trying to explain individual goals. Poole[Poo86] proved that g is explained from F and Δ, if and only if there exists a corresponding extension in which g is true. If it is possible to explain g and $\neg$g, then there are at least two distinct extensions.

If we define our theory comparator in terms of extensions, and not individual theories, we immediately satisfy the criteria for theory preference we presented above. By definition, an extension is consistent with the facts. Also, given an extension E, g $\in$ E, if and only if the subgoals of g are elements of E. If we replace definition 3 with:

Definition 4 *Given F, a set of facts, and Δ, a set of defaults, then (F,Δ) $\vdash$ g (g is predicted), if there exists an extension E such that $g \in E$, and, for every extension E' such that $g \notin E'$, $E \leq E'$.*

Note that if there are two maximally specific extensions, E_1 and E_2, then g is predicted if it is true in both extensions.

7.2 Preference among extensions

If we have two extensions, E_1 and E_2, then we need a semantic definition of what more specific means. Because extensions are uniquely determined by the set of defaults that are true in them, it seems that we want to compare the two sets of defaults. This coincides with our intuition that we prefer to use the most specific defaults.

There are two main problems with comparing extensions:

1. Determining which defaults are relevant to the comparison, and

2. Deciding which set of defaults is more specific.

The first problem is where Loui's and Poole's formalisms fail. In example 1, they include d_2 in the comparison, when it is irrelevant to determining whether $g_1 \wedge g_2$ is predicted. The two maximal default theories are:
$D_1 = \{d_1, d_2\}$
$D_2 = \{d_3, d_2\}$
Even though d_2 is not required to prove $\neg(g_1 \wedge g_2)$, d_2 is consistent with it. If we want to determine which of two extensions is more specific, we only want to consider the defaults on which they disagree. Clearly, neither d_1 nor d_3 is more specific, so we cannot predict g_1 or $g_1 \wedge g_2$. However, we are still able to predict g_2.

The second problem is a much more difficult one. Loui's and Poole's formalisms both agree that, given two contradictory defaults, one is preferred if its antecedents imply the antecedents of the other. In other words, the second is applicable (can be used in a proof), whenever the first is. In both of their formalisms, defaults could only be compared in the context of a particular proof tree.

At first glance, it seems that it is impossible to determine, for an extension, which facts are the antecedents and which are the consequents. However, this is not the case:

Definition 5 *Let D be a maximal default theory, corresponding to extension E. If* $d \in D$, *then* $(d \supset a_1 \vee a_2 \vee ... \vee a_n) \in F$. *If d is used in a proof of some element of E, then only one of* $a_1, a_2, ..., a_n$ *is true in E. Let* $a_n \in E$. *Then, the* antecedents *of d (written ant(d)) are* $\neg a_1 \wedge \neg a_2 \wedge ... \wedge \neg a_{n-1}$.

Clearly, at least one of $a_1, a_2, ..., a_n$ must be true if d is consistent. If more than one a_i is true, then d is consistent but is not necessary for the proof of any element in E. In this case, d is irrelevant to any conflict among potential conclusions.

This allows us to define what is meant by $E_1 \leq E_2$:

Definition 6 *Let* E_1 *and* E_2 *be extensions of F and* Δ, *with corresponding maximal default theories* D_1 *and* D_2. *Let* $D = D_1 \cap D_2$. *Let* $D_1' =$

$D_1 \backslash D and D_2' = D_2 \backslash D$. *Let* $S_1 = \bigcup\{ant(d) : d \in D_1'\}$ *and* $S_2 = \bigcup\{ant(d) : d \in D_2'\}$. $E_1 \leq E_2$, *if:*

1. $F_n \cup S_1 \cup D \models S_2$ *and*
2. $F_n \cup S_2 \cup D \not\models S_1$.

This definition of more specific deals with basic inheritance hierarchies and problem that arise out of conjunctions of defaults.

7.3 Examples Revisited

If we look at the examples presented earlier, we find that this definition deals with the problems discussed. For the expanded bird/emu case, there are two extensions with corresponding maximal default theories:

D_1 = { enf(fred), maybeflying(fred) }

D_2 = { bf(fred), maybeflying(fred) }

From definition 6, we get:

S_1 = { emu(fred) } and

S_2 = { bird(fred) }

which allows us to predict ¬flies(fred), but doesn't allows us to predict neither flying(fred) nor ¬flying(fred).

Similarly, in example 1 we are able to predict g_2, but neither g nor g_1. In example 2, there are nine possible extensions, but the preferred one is:

$E = \{a, b, \neg c, d, e, \neg f, g\}$

This corresponds to our intuitive notion of what are the preferred conclusions.

8 Conclusion

We have presented an analysis of the multiple extension problem and what properties a theory preference system ought to have. We then analysed the semantics of Poole and Loui in terms of these criteria, and presented a semantics that seems to remedy these problems. The major contributions of this paper are:

- a set of criteria that all useful theory preference systems must meet

- a semantics for comparing conflicting extensions and preferring the most specific one.

Preliminary work suggests that this handles the flaws found in other formalisms. In the future, we hope to develop a useful implementation of this semantics. The use of an ATMS[dK86], and its notion of nogood sets, appears to be a promising direction in which to look for such an implementation.

References

[dK86] Johan de Kleer. An assumption based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[ER83] David W. Etherington and Raymond Reiter. On inheritance hierarchies with exceptions. In *Proceedings of The National Conference for Artificial Intelligence*, pages 330–334, August 1983.

[GG86] S.D. Goodwin and R.G. Goebel. *Theory Preference based on Persistence.* Technical Report CS-86-34, Dept. of Computer Science, University of Waterloo, September 1986.

[Lou86] Ronald P. Loui. *Defeat Among Arguments: A System of Defeasible Inference.* Technical Report TR 190(Revised), University of Rochester, December 1986.

[PGA85] D.L. Poole, R.G. Goebel, and R. Aleliunas. Theorist: a logical reasoning system for defaults and diagnosis. In N.J. Cercone and G. McCalla, editors, *Knowledge Representation*, Springer-Verlag, New York, 1985. invited chapter, submitted Sept. 10.

[Poo85] D.L. Poole. On the comparison of theroies: preferring the most specific explanation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 144–147, August 1985.

[Poo86] David L. Poole. *Default Reasoning and Diagnosis as Theory Formation.* Technical Report CS-86-08, University of Waterloo, March 1986.

[RC81] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 270–276, August 1981.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, April 1980.

[Tou84] David S. Touretzky. Implicit ordering of defaults in inheritance systems. In *Proceedings of The National Conference for Artificial Intelligence*, pages 322–325, August 1984.

Adding conjunctions to inheritance graphs

Ross Bryant

LPAIG, Computer Science Department, University of Waterloo
May 11, 1987

ABSTRACT

Default reasoning systems such as Theorist frequently generate extensions which are mutually incompatible. Sometimes this is unavoidable and even desirable, but sometimes one extension is to be preferred on the grounds that the theory it is a consequence of is more specific. Touretzky's inheritance hierarchy provides a means of determining whether one theory is more specific than another. However, when conjunctions are added to an axiomatization, the method appears to be not applicable. A transformation on inheritance style graphs is suggested which replaces conjunctions with set intersections and associates defaults with these new nodes, so that the transformed graph can be analyzed by Touretzky's method.

1. Introduction: The Theorist Default Reasoning System

1.1. Explanations in Theorist

Theorist [Poole84] [Poole85a] is a default reasoning system in which a goal g is said to be "explainable" iff $F \cup D \models g$ and $F \cup D$ is consistent, where F is a set of facts and D is a set of instances of members of a set of defaults, Δ. Consider the following example:

Δ = {flies(x) ← bird(x),
¬ flies(x) ← emu(x)}

F = {bird(x) ← emu(x),
emu(tweety)}

g1 = flies(tweety)

g2 = ¬ flies(tweety)

Material implications which are members of F such as "bird(x) ← emu(x)" can be considered to be implicitly universally quantified, that is, equivalent to

$(\forall x)$(bird(x) ← emu(x))

Implications which are members of Δ, such as "flies(x) ← bird(x)", are not to be understood as universally quantified, since in general they are not meant to be true for all x. Instead they represent a relation of typicality, for example, birds typically fly, emus typically do not. Alternatively, they represent open wffs, instances of which we are prepared to assume given no evidence to the contrary.

Theorist can be used to provide "scientific" explanations of given observations. For example, if we are given as an observation g1 = flies(tweety), we can "explain" the observation with the theory D1 = {flies(tweety) ← bird(tweety)}.[1] On the other hand, if we are given as a observation g2 = ¬ flies(tweety), we can explain it with the theory D2 = {¬ flies(tweety) ← emu(tweety)}

Thus Theorist is well suited for generating explanations of observations from a given set of defaults. By putting the defaults in a special set Δ and then generating theories, Di, we get axiom sets $F \cup Di$, which are normal axiom sets of first order clausal logic (FOL). Thus, Theorist inherits the semantics of FOL , in particular, the soundness and completeness of its proof theory. No formal semantics for

[1] The term "theory" is meant here in the scientific sense, in which a theory explains or predicts a set of observations, as opposed to the logical sense of "theory" in which a theory is the deductive closure of a set of axioms.

members of Δ is provided except the restriction that instances of Δ making up D be consistent with F and each other. But there is no way given to determine whether an arbitrary member of Δ is "true" under a given interpretation I.

1.2. The problem of multiple extensions in Theorist

As long as we are using Theorist to explain given observations, there is no problem with this scheme. But when we reverse the process and ask what goals follow from a given axiomatization, interesting problems arise, as they do for all default reasoning systems. In terms of Theorist, we are asking the following:

$$F \cup \Delta \models ?$$

That is, what are we to mean by the logical consequences of a set of facts and defaults? (This corresponds to prediction of observations from a current hypothesis as opposed to generation of an hypothesis from observations. In the medical domain it includes prognosis or the prediction of the likely future course of a disease, as opposed to diagnosis of the disease itself.) In a way, this question is ill-formed for the reason that if we cannot determine the truth of wffs in Δ under interpretation I, we cannot determine the set of models of F ∪ Δ, and if we cannot do this it hardly makes sense to attach a notion of logical consequence to F ∪ Δ. However, we can give a definition of the set of "extensions" of F ∪ Δ as follows. Define D to be the set of all theories Di such that F ∪ Di is consistent. Associate with each Di an "extension" Ei where Ei is a set of wffs satisfying the following conditions:

(1) $F \cup Di \subseteq Ei$

(2) $Ei = Th(Ei)$

The first condition makes F ∪ Di the axioms of Ei, and the second condition means that Ei is closed under theoremhood. The "extensions" of F ∪ Δ will be the set of all Ei. Notice that the Ei's are really just deductive closures and as such could be called "theories " in the logical sense, but the term "extension" is commonly used since [Reiter 80] for closures of default systems, and I will use the same terminology. (Reiter's extensions are maximal sets, but I do not believe this is a significant issue here.) A given axiomatization in Theorist will normally have more than one extension (thus "multiple extensions") and in some cases there will be extensions which are mutually inconsistent. There are situations, however, in which we have reasons for "preferring" one extension over another. The above "tweety" example is one of these. Given the information in F and Δ, it seems preferable to conclude that tweety does not fly because that goal follows from a theory which uses a default rule with "more specific" information, ie., it reasons from the fact that tweety is an emu, and emus are a subset of birds. In general, the "problem" of multiple extensions is that of (1) providing a clear criterion for preferring one theory over another, and (2) providing as good a rationale or justification for the criterion as possible.

1.3. The intuition behind "more specific"

In the "tweety" example, the word "specific" seems to carry the connotation of a species which is a member of a genus of organisms. Removing the biological overtones, "more specific" theories are theories about subsets of sets. For example, when we say that "birds fly" is a default as opposed to a fact, we are saying that typically birds fly but possibly some birds do not. We note the fact that all emus are birds, and that "emus do not fly" is a default. So the instances of birds which do not fly are not "evenly" spread over the various species of birds. The emus default tells us that most emus are deviant with respect to this typical characteristic of birds. Notice that there is nothing "contradictory" about this situation even though Theorist can be used to "explain" goals which are inconsistent. In fact, the preference for the prediction that ¬ flies(tweety) is almost like a conditional probability statement:

$$P[\neg \text{flies(tweety)} \mid \text{emu(tweety}] > P[\text{flies(tweety)} \mid \text{emu(tweety)}]$$

Theorist alone gives us no grounds for rejecting "flies(tweety)", and in fact we would not want to since it is a possible observation. However, we can try to provide a preference criterion so that we can determine when one theory can be preferred over another. Since standard Theorist has no formal semantics for Δ, we will not be able to prove that our criterion is "correct" because there is no standard it must measure up to except our own intuitions concerning reasoning about subsets and the defaults associated with them.

1.4. A more general definition of "more specific"

In [Poole 85b] a general definition of "more specific" is given which is applicable to any type of formula not just implications with unary predicates such as the "tweety" example. In fact, the definition applies equally to purely propositional formulas. The definition makes use of a subset relation on the sets of models of subsets of axioms. Thus, while it is more general, it loses the detail of the intuition outlined above.

2. Inheritance Hierarchies

[Touretzky84] represents the subset relation in a type of graph called an "inheritance hierarchy". (See Figure 1 for an inheritance hierarchy representing the "tweety" example.) The idea behind the name is that individuals can inherit properties (membership in various sets) from nodes higher up the graph. An arrow from p to q represents the default "$q(x) \leftarrow p(x)$". A hatched arrow from p to q represents the default "$\neg\, q(x) \leftarrow p(x)$". In order to represent Theorist axiomatizations, I use a double arrow from p to q to represent the fact "$(\forall x)(q(x) \leftarrow p(x))$". (Touretzky uses only defaults in order to be consistent with FRL and NETL, the systems he is attempting to model.)[2] I also use a squiggly line to represent set membership (such as p(a)) so that it is distinguished from set inclusion. The inheritance hierarchy provides a clear way of visualizing the relationships in a default axiomatization. In the example, it is clear that D2 is a more specific theory than D1 because it emanates from a node in the hierarchy which is lower than D1. In fact, this in a nutshell is Touretzky's criterion for "more specific". (For details of the algorithm, see [Touretzky 84].)

2.1. Inheritance hierarchies and FOL semantics

It should be pointed out that representing material implication with the set inclusion relation is essentially using regular (Tarskian) FOL semantics. When an interpretation I is given to a set of wffs in first order logic, individual symbols are mapped to elements of a domain D, n-ary function symbols are mapped to n-ary functions on D and n-ary predicate symbols are mapped to n-ary relations on D. From this basic interpretation of the constants in the language, the truth of any atomic formulas is determined. The truth of molecular formulas built out of atomic formulas with sentential connectives is defined recursively with the familiar truth tables for the connectives. In addition, the truth values of formulas which contain universal and existential quantifiers are determined by considering the truth value of the wff for elements of domain D. A unary predicate symbol, p, will be mapped to a unary relation, say I(p), which can be regarded as the set of elements of D which defines the relation. Thus $I(p) \subseteq D$, and it is really I(p) which is represented in the inheritance hierarchy. That is, the inheritance hierarchy is in a sense the **meaning** of the axiomatization it represents, at least as far as factual links are concerned.[3] The fact "$(\forall x)(q(x) \leftarrow p(x))$" is true precisely when $I(p) \subseteq I(q)$.

Although we have no definition of truth for elements of Δ, a natural extension is that the default "$q(x) \leftarrow p(x)$" is true iff $\text{Card}(I(p) \cap I(q)) > \text{Card}(I(p) \cap \text{complement}(I(q)))$. That is, more p's are q's than are not-q's. Or, expressed in a slightly stronger way, **most** p's are q's. Whether or not we wish to provide a formal semantics for members of Δ,[4] it seems that we must have something like this in mind if we are to have a preference for more specific theories. Notice that this definition of truth applies only to those members of Δ which are statements of typicality, such as "birds fly". Other defaults such as "has_cold(x)" which might be used in a diagnostic system to explain why a patient coughs are certainly not meant to be true only if most people have colds. The "truth" of these sorts of defaults seems to be the very weak condition of mere possibility. In the above example, we mean that it is possible that a person has a cold.

---

[2] See [Etherington87]for inheritance hierarchies with a combination of facts and defaults.

[3] In [Etherington87] it is claimed that logic provides a semantics for inheritance hierarchies. This is hard to understand unless it means FOL semantics as outlined above. In any case, Reiter's default logic has no semantics for default rules of inference. They function as "meta rules" which allow us to determine what we are prepared to assert as non-logical axioms.

[4] In ordinary reasoning we are prepared to say that defaults such as "birds typically fly" or "most birds fly" are true, whereas ones like "birds typically don't fly" are false.

2.2. The Republican-Quaker Example

It was observed at the outset that a default axiomatization may have inconsistent extensions, one of which is preferred. It is also possible to have inconsistent extensions neither of which is preferable to the other. For example, consider the following:

Δ = {pacifist(x) ← quaker(x),

¬ pacifist(x) ← pro-defense(x),

pro-defense(x) ← republican(x)}

F = {quaker(Nixon), republican(Nixon)}

g1 = pacifist(Nixon)

g2 = ¬ pacifist(Nixon)

(See Figure 2 for the inheritance hierarchy for this example.) We can explain g1 with D1 = {pacifist(Nixon) ← quaker(Nixon)} but we can equally explain g2 = ¬ g1 with D2 = {¬ pacifist(Nixon) ← pro-defense(Nixon), pro-defense(Nixon) ← republican(Nixon)} This example seems genuinely ambiguous - we have no preference for either theory. The inheritance graph shows that neither default emanates from a node which is represented as a subset of a node from which the other default emanates. Notice that we do not need an example this complicated to demonstrate ambiguity. The following example will suffice:

Δ = {p(x), ¬ p(x)}

F = {}

D1 = {p(a)} will explain p(a), D2 = {¬ p(a)} will explain ¬ p(a), and neither theory is more specific. (The purpose of the extra links in the Republican-Quaker example is to point out the deficiencies of FRL and NETL. FRL does not even notice that the two extensions are inconsistent, while NETL prefers (incorrectly) pacifist(Nixon) because the proof has fewer links. [Touretzky84]) The result is that inheritance hierarchies can be used to determine whether one theory is more specific than another or not, at least in simple cases.

3. Axiomatizations with conjunction

There are examples of default axiomatizations involving conjunctions which do not appear to be analyzable by inheritance hierarchies. In some of these examples it is not even clear to us which of two incompatible theories we ought to prefer, if either.

3.1. The Feathers example

Consider the following example[5] in which there is a conjunction in the goals to be explained.

Δ = {flies(x) ← bird(x),

¬ flies(x) ← emu(x)}

F = {bird(x) ← emu(x),

feathers(x) ← bird(x)

emu(tweety)}

g1 = flies(tweety) ∧ feathers(tweety)

g2 = ¬ flies(tweety) ∧ feathers(tweety)

This example cannot be represented by a normal inheritance hierarchy so I add the convention of representing conjunction with arrows joined by an arc. (See Figure 3 for the inheritance style graph.) The goal g1 can be explained with the theory D1 = {flies(tweety) ← bird(tweety)} and g2 can be explained with the theory D2 = {¬ flies(tweety) ← emu(tweety)}. It seems that we ought to prefer g2 but the inheritance graph is less clear than in the first "tweety" example. D2 is still more specific than D1

[5] This example was suggested by David Poole

but the goals do not depend on these defaults alone.

3.2. Three isomorphic examples

The next three examples are syntactically isomorphic and logically equivalent, but our intuitions concerning specificity are inconsistent. The examples involve conjunctions in the antecedents of implications and also multiple facts about the same individual.

3.2.1. The live-bird example[6]

Δ = {flies(x) ← (bird(x) ∧ alive(x)),

¬ flies(x) ← emu(x)}

F = {bird(x) ← emu(x),

emu(tweety),

alive(tweety)}

(See Figure 4.) We can explain flies(tweety) with D1 = {flies(tweety) ← (bird(tweety) ∧ alive(tweety))}, but we can also explain ¬ flies(tweety) with D2 = {¬ flies(tweety ← emu(tweety)}. By adding conjunctions, we again cannot apply Touretzky's criterion. Our preference seems to be that an alive emu does not typically fly and that D2 is still the more specific theory.

3.2.2. The trained-tiger example[7]

Δ = {does-tricks(x) ← (animal(x) ∧ trained(x)),

¬ does-tricks(x) ← tiger(x)}

F = {animal(x) ← tiger(x),

tiger(tony),

trained(tony)}

(See Figure 5.) In this case we prefer does-tricks(tony) because tricks are what trained tigers typically do. But this is the opposite preference from the preceding example, which has the same syntactic structure. Clearly, our preferences are being conditioned by something other than the axiomatization given. In the alive-bird example it is something like biological knowledge which tells us that aliveness is a property which if anything enhances the applicability of the usual defaults which apply to a species (unlike deadness). In the trained-tiger example, the normal default is over-ridden because of the meanings of "trained" and "does-tricks". The next example clarifies the situation.

3.2.3. The Rembrandt-landscape example

Δ = {like(x) ← (painting(x) ∧ rembrandt(x)),

¬ like(x) ← landscape(x)}

F = {painting(x) ← landscape(x),

landscape(a),

rembrandt(a)}

(See Figure 6.) In this example we have an individual who likes Rembrandt's paintings but not landscapes, and we are wondering whether he likes an object 'a' which is a Rembrandt landscape. In this example, which is isomorphic to the previous two, we clearly have no preference for like or not-like. It could be that Rembrandt's landscapes are precisely those which are the exception to the person's general dislike of landscapes. On the other hand it may be that Rembrandt's landscapes are the exception to the rule that Rembrandt's paintings are liked. Or there may be any range of attitudes in between so that no default applies.[8] We do not have enough information and there are no other clues we can "read into" the

[6] This example was suggested by Bruce Kirby

[7] The next two examples were suggested by Eric Neufeld.

[8] The lack of information about the intersection of landscapes and Rembrandt's painting was pointed out by Eric

situation as with the previous two examples.

3.2.4. The significance of the three isomorphic examples

What the three examples show is that we need a better criterion of "more specific" than simply looking at examples with intended interpretatons, registering an intuition as to which if any of two incompatible theories is more specific, and then using this to select or tune an algorithm. One strategy is to replace ordinary language predicates with arbitrary variables and then ask ourselves whether one theory is more specific than another. (See Figure 7 for an example of this type which is isomorphic to the previous three.) Representing the structure of an axiomatization in this way prevents us from reading anything into it. However, it is not obvious from the inheritance-style graph representing this axiomatization whether one theory is in fact more specific than the other or whether they are equally specific. The reason for this is that the introduction of conjuctions implicitly defines intersections of sets in the graph, and it is these intersections which are needed to establish the correct inheritance structure.

4. An inheritance graph transformation

A conjunction such as "$p(x) \wedge q(x)$" occurring in a formula can be seen as defining a set $I(p) \cap I(q)$ where $I(p)$ and $I(q)$ are the sets defining the relations denoted by predicate symbols p and q under interpretation I. That is, the conjunction will be true iff x is a member of the intersection. In a formula such as

$$r(x) \leftarrow (p(x) \wedge q(x))$$

the corresponding link in an inheritance graph can be associated with a node representing $I(p) \cap I(q)$. So what is needed is a transformation of the inheritance-style graph to a genuine inheritance graph which includes the necessary set intersections and redefined default and factual links. I use the Rembrandt-landscape example to suggest[9] how this can be done. The idea is to start at the bottom of the inheritance-style graph and work upwards. I will describe the transformation as a transformation on the axiomatization itself. (See Figure 10 for the result we are ultimately aiming for.)

There are two atomic facts about the individual denoted by "a", and they are equivalent to the conjunction:

$$landscape(a) \wedge rembrandt(a)$$

We introduce the intersection of I(landscape) and I(rembrandt) with the wff

$$landscape.rembrandt(x) \leftarrow (landscape(x) \wedge rembrandt(x))$$

(The notation is meant to indicate that $I(landscape.rembrandt) = I(landscape) \cap I(rembrandt)$.) We add to F wffs for the links to landscape and rembrandt from their intersection:

$$landscape(x) \leftarrow landscape.rembrandt(x)$$

$$rembrandt(x) \leftarrow landscape.rembrandt(x)$$

Now we add the fact: landscape.rembrandt(a) and remove the original facts:

$$rembrandt(a)$$

$$landscape(a)$$

(Removing these facts forces inferences to go through the intersection landscape.rembrandt.) At this point, if there were any more multiple facts about a single individual, we would repeat the process. See Figure 8 for the change so far.

Proceed up the graph (i.e., trace the implications in $F \cup \Delta$) until a conjunction in an antecedent is found.[10] In this example the only other conjunction appears in the default:

Neufeld.

[9] I have not yet worked out a precise algorithm for this.

[10] The main problem I anticipate is ruling out possible unwanted interactions between transformations done to nodes which are at the same level in the graph.

like(x) ← (painting(x) ∧ rembrandt(x))

Define the intersection of I(painting) and I(rembrandt) with the wff:

painting.rembrandt(x) ← (painting(x) ∧ rembrandt(x))

Add the following wffs to F:

painting(x) ← painting.rembrandt(x)

rembrandt(x) ← painting.rembrandt(x)

This establishes I(painting.rembrandt) as the intersection of I(painting) and I(rembrandt). Next we remove the default and replace it with an equivalent default associated with the intersection:

like(x) ← painting.rembrandt(x)

This forces inferences to go through the intersection painting.rembrandt. This results in the graph shown in Figure 9.

The only complicated step is determining whether we need to add links with the new node painting.rembrandt as destination. To do this, we consider each path back from painting in turn and all paths back from rembrandt to check whether there is a common node. In the simple example shown there is a common node (landscape.rembrandt) from which there are paths (consisting entirely of facts in this case) to painting and rembrandt. Therefore we add the new fact:

painting.rembrandt(x) ← landscape.rembrandt(x)

This results in the graph shown in Figure 10. Notice that this is a proper inheritance hierarchy now that all conjunctions have been removed. The new axiomatization we have produced is:

T(Δ) = {like(x) ← painting.rembrandt(x)

¬ like(x) ← landscape(x)}

T(F) = {painting(x) ← landscape(x),

painting(x) ← painting.rembrandt(x),

rembrandt(x) ← painting.rembrandt(x),

landscape(x) ← landscape.rembrandt(x),

rembrandt(x) ← landscape.rembrandt(x),

landscape.rembrandt(a)}

We can explain likes(a) with D1 = {likes(a) ← painting.rembrandt(a)} and we can also explain ¬ likes(a) with D2 = {¬ likes(a) ← landscape(a)} as we did before. But now all information has been represented explicitly as set inclusion, and we can see that we have an axiomatization which is essentially similar to the Republican-Quaker example. Both defaults are associated with subsets of painting so neither theory is more specific than the other. This is the result we were led to by a consideration of the Rembrandt-landscape example. The transformed graph also makes it more obvious that the information we would need in order to have a preference is knowledge about the intersection of 'landscapes' and 'painting.rembrandt'. If this information could be added to the axiomatization as a fact or a rule, it would then be more specific than either of the two existing defaults. This is shown in Figure 12. This illustrates that we can prefer one theory over another even if they have the same goal, not just if they have inconsistent goals. If the new information is a default, it must agree with either D1 or D2.[11]

---

[11] The Rembrandt-landscape example illustrates a possible application of a default reasoning system. Consider an art dealer who buys art objects on speculation with the hope of selling them to clients. A typical client's taste can be given a Theorist axiomatization in terms of facts and defaults. It now becomes important for the art dealer to know for any potential purchase whether a given client will probably like it. Therefore his system, automated or otherwise, must be able to detect inconsistent predictions and determine whether there is a preference for one or the other. In cases where ambiguity is detected he will want to upgrade his knowledge by consulting with the client in question. As noted before, there are a variety of possibilities in this case, including the situation in which the client has no pattern of preference for Rembrandt landscapes. In this case the representation of his preferences will remain ambiguous on this point.

4.1. The Feathers example revisited

If we apply the same transformation idea to the "feathers" example, we get the transformed axiomatization:

$T(\Delta) = \{$feathers.flies(x) $\leftarrow$ bird(x),
feathers.$\neg$ flies(x) $\leftarrow$ emu(x),
flies(x) $\leftarrow$ bird(x),
$\neg$ flies(x) $\leftarrow$ emu(x)$\}$

$T(F) = \{$g1 $\leftarrow$ feather.flies(x),
g2 $\leftarrow$ feathers.$\neg$ flies(x),
feathers(x) $\leftarrow$ bird(x),
bird(x) $\leftarrow$ emu(x),
emu(tweety)$\}$

Notationally, "feathers.$\neg$ flies(x)" means "x has feathers and does not fly". See Figure 13 for the inheritance graph.[12] We can explain g1 with the theory D1 = {feather.flies(x) $\leftarrow$ bird(x)}, but we can explain g2 with the theory D2 = {feathers.$\neg$ flies(tweety) $\leftarrow$ emu(tweety)}, and D2 is more specific than D1. Therefore we achieve the desired result that emus typically have feathers but don't fly.

4.2. Complexity issues

If there are n distinct predicate symbols in an axiomatization, there will be 2n sets associated (allowing for complements of sets to correspond to negation.) In the worst case, the number of nodes will be 2**2n representing all possible intersections. The number of arcs needed to define the set inclusion relation in the worst case will be worse than exponential. However, in a given situation, we need not make the complete graph explicit, just the nodes which correspond to conjunctions in the axiomatization. As far as the transformation itself is concerned, the only potentially costly search is for new links terminating at added nodes.

On the other hand, the transformation I have described might not be the most efficient way of determining specificity, but it seems to be a natural way to **define** it. If it is correct, then any algorithm which determines whether a theory is more specific than another must be able to produce the same results.

5. Summary

A possible method for determining whether a theory is or is not more specific than another in cases involving conjunction has been outlined. It has been shown capable of solving simple problems in axiomatizations which involve only unary predicates.

6. Future Work

The next step is to produce a general description of the transformation and to show that it produces unique results. It will probably be necessary to restrict the allowable wffs to single implications, but allow conjunctions in antecedents and goals as well as having multiple facts about the same individual.[13]

Another problem being examined is to extend the transformation to handle conjunctive goals involving different individuals, for example:

g = flies(tweety) $\wedge$ flies(polly)

---

[12] In this example, when new links are formed which have the intersections as a destination, the paths back to the common nodes ('bird' and 'emu') consist of a combination of facts and defaults. Therefore, the new links can be defaults at best. We do seem to lose some information, however. In the new link, it is not clear whether the default is associated with feathers or flying or both.

[13] Disjunction is easily represented as links leaving a node.

Reasoning about n individuals seems to suggest n-ary relations in which case the nodes to be added will correspond to sets of n-tuples. Examples studied suggest this might work, but the graphs become very complicated to generate.

Examples involving propositional logic wffs seem amenable to the method described, the problem being that of attaching a meaning to more specific in the absence of predicates which can be associated with sets on a domain. The definition in [Poole85b] might be applicable.

7. Acknowledgements

I would like to thank the following people for their advice and suggestions: Eric Neufeld, David Poole, Denis Gagne, Randy Goebel, Bruce Kirby. I hope I didn't forget anyone!

8. Bibliography

[Etherington83]
Etherington D.W., and Reiter, R., "On inheritance hierarchies with exceptions", *Proceedings, AAAI-83*, pp 104-108.

[Etherington87]
Etherington, D.W., "Formalizing Nonmonotonic Reasoning Systems", *Artificial Intelligence* 1987, pp41-85.

[Moore83]
Moore,R.C., "Semantical Considerations on Nonmonotonic Logic", *IJCAI-83*, pp. 272-279.

[Poole84]
Poole, D.L., "A Logical System for Default Reasoning", *AAAI workshop on Nonmonotonic Reasoning*, NY Oct 1984, pp 373-384.

[Poole85a]
Poole D.L., Goebel R.G., and Aleliunas, R., "Theorist: a logical reasoning system for defaults and diagnosis", invited to appear in N.Cercone and G. McCalla (Eds.) *Knowledge Representation*.

[Poole85b]
Poole D.L., "On the Comparison of Theories: Preferring the Most Specific Explanation", *Proc. AAAI-85*.

[Reiter80]
Reiter, R., "A logic for default reasoning", *Artificial Intelligence*, Vol 13, pp 81-132.

[Touretzky84]
Touretzky, David, "Implicit Ordering of Defaults in Inheritance Systems", *Proceedings AAAI-84*, pp. 322-325.

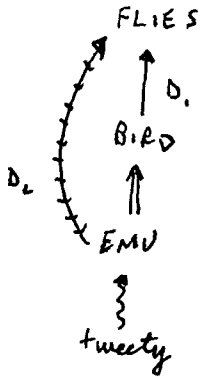


Figure 1

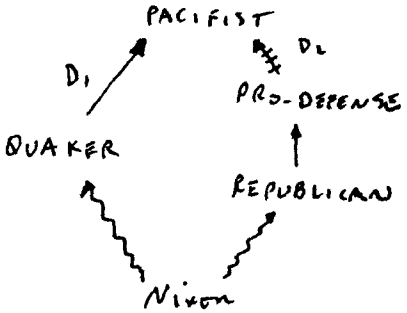


Figure 2

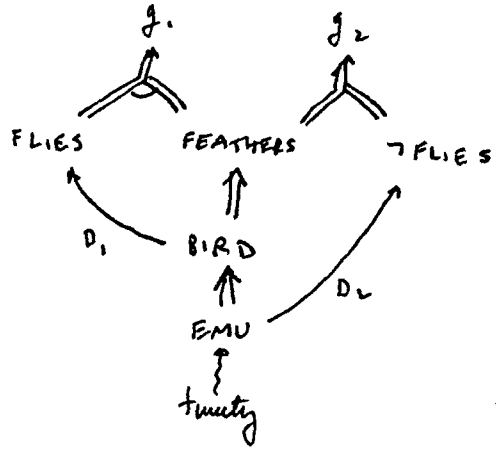


Figure 3

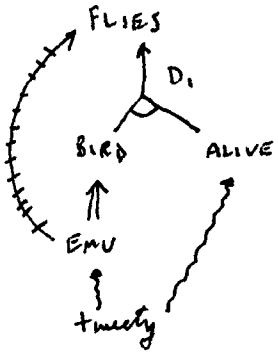


Figure 4

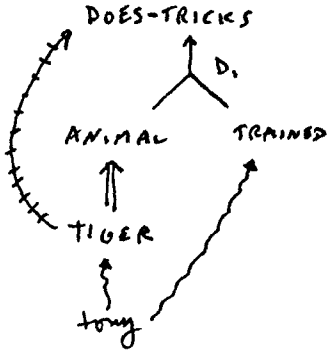


Figure 5

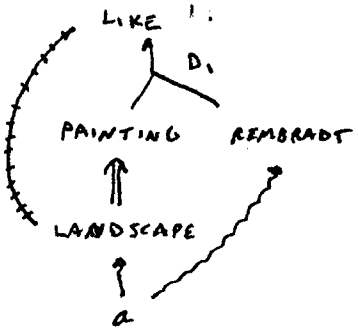


Figure 6

Figure 7

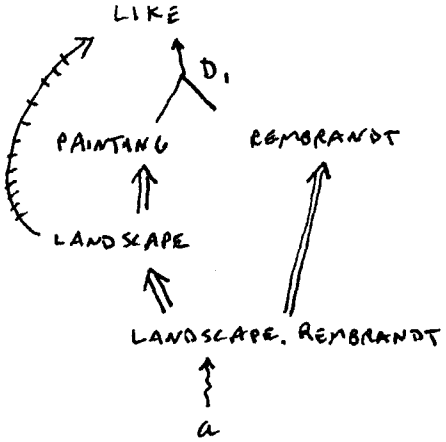


Figure 8

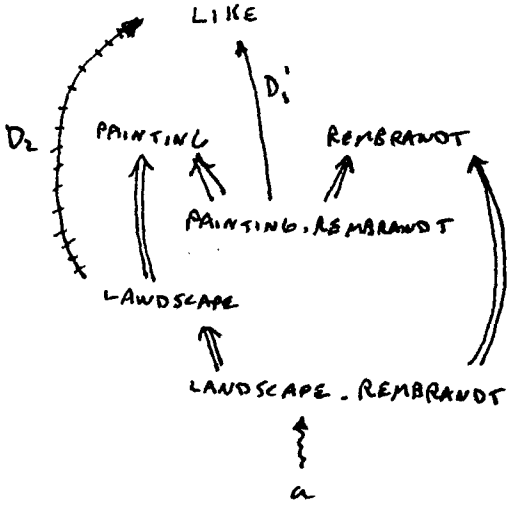


Figure 9

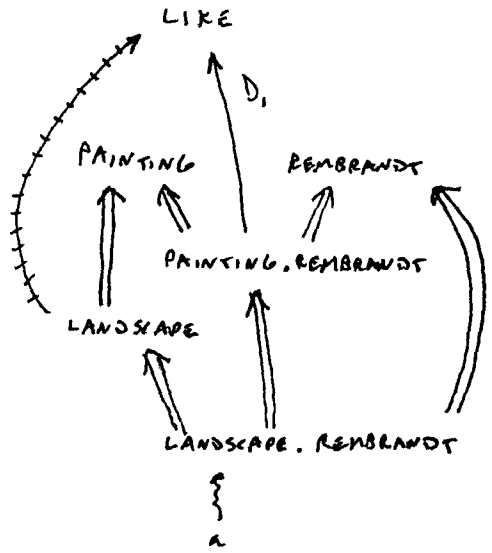


Figure 10

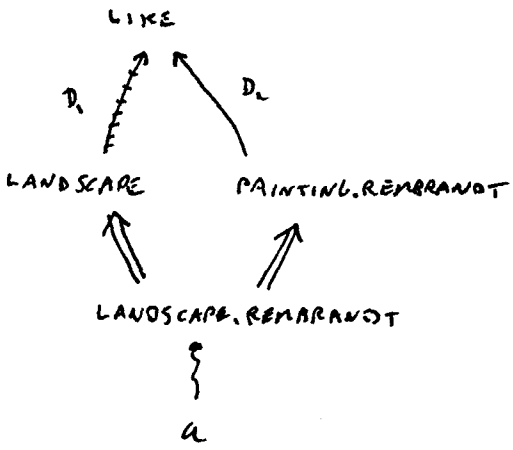


Figure 11

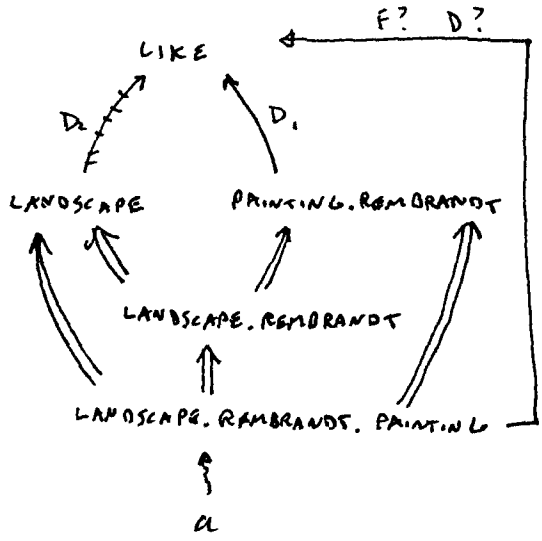


Figure 12

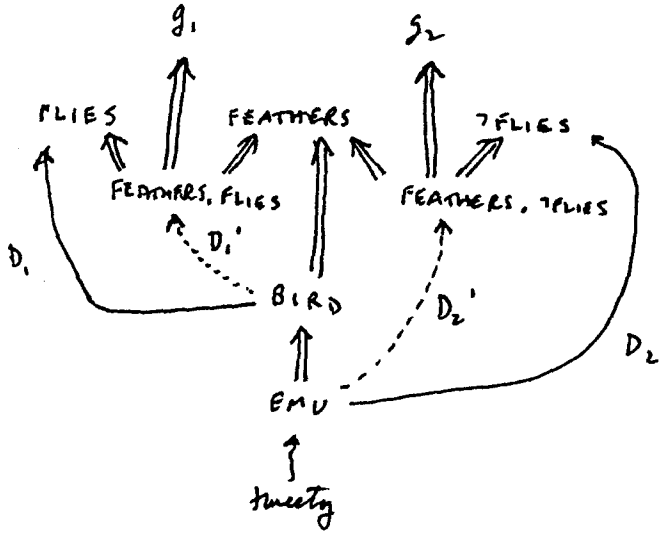


Figure 13

CS786
Term paper

Theorist, Theory Preference, and Inheritance systems

Richard Lemieux, Winter 1987

Introduction

Touretzky (1984b) has provided a formalization of inheritance systems that can handle correctly the precedence of defaults for a large number of usual cases. Poole (1985) proposes a semantics and a discriminant based on that semantics in order to select the most specific theory from the solutions generated by Theorist. This paper presents a review of *Theorist* and *Theory Preference* in the light of Touretzky's *Inheritance System* in order to develop an understanding of some problems that are met whith the current implementation of Theorist. We propose additional concepts to be introduced in the framework of Theorist that would provide a basis for a new implementation.

A first problem is that the theories generated by the current Theorist implementation sometimes appear unnatural in the sense that they appear unrelated with the set of facts; this concept will become more apparent later when we discuss defective theories. The other problem is that while Theory Preference handles correctly some isolated sets of facts and defaults, it seems to fail when the initial set of facts and defaults is embedded into a larger context. A related problem is that the definition of Theorist is not very explicit both on what the facts are and what the defaults are and we are led to think that by moving clauses from the set of defaults to the set of facts or vice versa Theorist will provide different answers, which is not a comfortable situation. This problem of deciding what the facts are is also apparent when trying to apply Theory Preference. The study of the axiomatization of Touretzky provides some clues on what the missing links are in the Theorist framework.

Our initial approach to handle those problems was to introduce an additional structure - the *critic* - in order to make some sense out of the theories generated by Theorist and to relate the theories to some model of the user of the information. Another concern we had with Theorist was that it might be inefficient at handling large databases. From our work, it came out however that many of the problems mentioned above could be fixed by remodeling Theorist itself rather than by introducing the additional concept of the critic. We also developed the intuition that the basic computational requirements ought not to be as bad as it was first thought and that a semi-parallel approach could be devised to handle the computations.

Background

The problem of implicitly making the closed world assumption for a set of logical predicates and of restricting the logic so that reasonable and predictable answers can be generated has given rise to the formalisms for default reasoning. Three formalisms have been under scrutiny during the recent years: nonmonotonic logic, circumscription and default logic.

Reiter (1980) introduced the concept of an extension; it is a minimum fixed point under theoremhood and it features a closed-world assumption for all defaulted predicates. Etherington and Reiter (1983) have provided a procedure to find an extension for acyclic hierarchies with exceptions, and Etherington (1983) has proved that the procedure always converges on an extension for such a

hierarchy. Lifschitz (1985) has shown that circumscription and the closed world assumption are equivalent if we assume the domain closure and the uniqueness of names. On another hand, Imielinski (1985) has proved that ordered combinations of seminormal defaults without prerequisites can be translated into circumscription. He also showed that no mechanism based on the partial order relation can capture defaults with prerequisites.

Etherington (1987) proves that for finite, ordered, *network theories*, the procedure given (Etherington and Reiter 1983) always converges on an extension.

Definition (Network theory). A default theory $\Delta = (D,W)$ is a network theory iff it satisfies the following conditions:

(1) W contains only:
  (a) literals (i.e., atomic formulae or their negations), and
  (b) disjuncts of the form $(a \vee b)$ where a and b are literals.

(2) D contains only normal and seminormal defaults of the form: $a:b/b$ or $a:b \wedge g_1 \wedge ... \wedge g_n/b$ where a, b, and g_i are literals. (Etherington 1987)

Touretzky (1984b) has provided a formalization of inheritance systems. It looks like as if he was computing circumscription. The class/property systems are included as a special case.

Poole(1985) discusses the application of the principle of the most specific theory as a criterion for theory preference. Theories are defined as part of the framework of Theorist (Poole et al. 1986, Poole 1986); given a set of *facts*, a set of *defaults*, and an *observation*, Theorist generates the set of all consistent combinations of instances of facts and defaults that can explain the observation. Each combination of instances of defaults that is consistent is called a *theory*. No instance of a default may contradict a fact.

The representation of facts and defaults

Facts and defaults are represented in Theorist in a clausal form. For example if we want to represent the facts that Tweety is a bird, that Tweety does not fly and the default that birds fly, we would write:

```
default d(1): fly(X) <- bird(X);
fact bird(tweety);
fact n(fly(tweety));
```

or

```
default d(1): fly <- bird;
fact bird <- tweety;
fact n(fly) <- tweety;
fact tweety;
```

Using the Touretzky mode of representation, we would write,

```
<+tweety, -fly>
<+tweety, +bird>
<+bird, +fly>
```

where the differentiation between facts and defaults has been lost. Those clauses have a graphical interpretation where each pair is represented by a directed arc. Arcs leading to a negative conclusion are crosshatched.

The mathematics of inheritance systems (Touretzky 1984b)

Touretzky has provided a formalization of inheritance systems based on the notion of an inferential distance. The idea is to get rid of some of the paths which may lead to inconsistent nodes. Basically one starts with a set of links in a directed graph (see the Clyde example below and Figs. 1 and 2). The nodes may denote individuals, predicates, or properties. Links ending with a negative token have the effect of negating the predicate or the property.

A problem is first expressed in logical sentences as a set of ordered pairs (see the Clyde example). The transitive closure of this set is then computed with the constraints to inheritance which are imposed under the axioms of inferential distance ordering. The transitive closure (under inheritance) may not be unique and then there are many closure sets. Each set correspond to a possible extension. Inference distance provides a clue on how to choose between two different extensions based on contradicting conclusions if there is a bridge (preclusion) between the conclusions.

The inferential distance concept is summarized by Touretzky (1984b) as follows: "'A may view B as a subclass of C iff A has an inference path *via* B to C, and not vice versa.' Suppose we are trying to find out if A has property P. If A inherits from B (which has property P) and A also inherits from C (which has property ~P), what conclusion should we reach about A? The inferential distance ordering says: if A has an inheritance path via B to C and not vice versa, then conclude P; if A has an inheritance path via C to B and not vice versa, then conclude ~P; otherwise report an ambiguity."

Now, let introduce some notation from Touretzky (1984b).

- Γ The given set of facts and defaults.
- Φ An expansion of Γ. Φ includes all of Γ and chains of inference that are built from elements of Γ. There may be many expansions of Γ such that $\Gamma \models \Phi_1$, $\Gamma \models \Phi_2$, $\Phi_1 \not\models \Phi_2$, and $\Phi_2 \not\models \Phi_1$.

Using the language of Touretzky, *conclusions* from a set of *sequences* Γ are obtained by first generating the *grounded expansions* of Γ (Φ's; see also the reference page) and then extracting the first and last element of each sequence to form a conjunction of pairs: the *conclusion set* - $C(\Phi)$. The following example is taken from Touretzky (1984b).

$\Gamma = \{$	<+clyde, +elephant>/	Clyde is an elephant.
	<+elephant, +gray.thing>/	Elephants are gray.
	<+gray.thing, +drab.thing>/	Gray things are drab.
	<+royal.elephant, +elephant>/	Royal elephants are elephants.
	<+royal.elephant, -gray.thing>	Royal elephants are not gray.
	<+clyde, +royal.elephant>	Clyde is a royal elephant. }

Γ has a single grounded expansion - all sequences in Φ are inheritable from Γ and Φ contains every sequence inheritable in Φ.

$\Phi = \Gamma \cup \{$ <+elephant, +gray.thing, +drab.thing>
<+clyde, +royal.elephant, +elephant>
<+clyde, +royal.elephant, -gray.thing> }

The conclusion set drawn from the expansion is the set of pairs:

$C(\Phi) = \Gamma \cup$ <+elephant, +drab.thing> $\cup$ <+clyde, -gray.thing>

Note that <+elephant, +drab.thing> is an intensional statement here since it refers to no specific instance of an elephant. Now, under the definition of inheritability, Φ precludes

<+clyde, +elephant, +gray.thing>

because Φ contains the two sequences,

<+royal.elephant, -gray.thing>
<+clyde, +royal.elephant, +elephant>

It precludes any sequence ending as *+gray.thing* since *+royal.elephant* is an *intermediary* in Φ. The intermediate step <+clyde, +royal.elephant, +elephant> appears to be critical to the decision of precluding <+clyde, +elephant, +gray.thing>. As we will see later such intermediate steps are left over by Theorist and are not available later when trying to compute the most specific theory. In this specific case, Theory Preference would make use of the ground default <+royal.elephant, +elephant> to correctly generate a decision; we will attempt to demonstrate that this is not enough in the general case since the default we need may not be given but have to be derived.

The most specific theory (Poole 1985)

Poole (1985) wants his definition of more specific to be a semantic rather than syntactic definition. The idea is to find a combination of the facts such that one theory explains some observation while another theory cannot explain another observation and still the first theory cannot explain the second observation. The first theory is then said to be *more general* than the second theory. A theory is said to be *more specific* than another theory if it can not be found to be more general. Theory preference favours the *most specific theory*, which results in choosing the theory which can accomodate an additional fact and remain consistent - this is different than choosing the theory that accomodates the largest number of facts.

The discriminant introduced by Poole (1985) compares solution pairs S1 = <D1,g1> and S2 = <D2,g2> where the D's are theories and g's are observations. Facts are partitioned in two subsets: the necessary facts (Fn) and the contingent facts (Fc). The solution S1 is said to be more general than S2 if there is a possible fact Fp, such that

$Fp \cup D1 \cup Fn \models g1$ and
$Fp \cup D2 \cup Fn \not\models g2$ and

$Fp \cup D2 \cup Fn \not\models g2$

We are going to use a notation which differs in some respects from the one used by Poole (1985) in order to make the comparison with the inferential distance of Touretzky. For instance, we are not making the distinction between necessary facts and contingent facts. Rather we adopt the approach of Touretzky where all relations are defaults. The only factual entities are the individuals. We will not be capable of applying Theory Preference in its original form. We are rather going to consider Fp as representing a set of facts and we will be looking for the unique fact that allows us to discriminate between the theories. Fn is considered to be empty. The equivalence between our approach and the definition given by Poole becomes clear if we can consider that the solutions S1 and S2 belong to different extensions and that a single (set of) facts is necessary to discriminate between the theories. Note that the fact we are we are considering here may be a conclusion taken from a grounded expansion; it does not have to be a given fact or default. Such a case would occur in the Clyde example (Fig. 1) if <+royal.elephant, +elephant> had to be deduced from more elementary facts such as:

<+royal.elephant, +tamed.elephant> and
<+tamed.elephant, +elephant>

We are now going to apply our modified form of Theory Preference to the problem of the elephant. Theories are first computed by Theorist (Fig. 3) and then we find the most specific theory. What is to be explained is that *gray* is a consistent conclusion given the defaults and the fact that *clyde* is an individual. Theorist can generate theories for both *gray* and *not gray*. The results from Theorist are, using fi for either a fact or a default, and Ti for a theory from Theorist:

f1: elephant <- clyde;
T1: gray <- elephant;

f2: royal-e <- clyde;
f3: elephant <- royal-e;
T1: gray <- elephant;

f2: royal-e <- clyde;
T2: n(gray) <- royal-e;

Now let us compute the discriminant for many combinations of fi, where we compare T1 explaining *gray* and T2 explaining *not gray*.

f1	T1	$\models$	gray	
f1	T2	$\not\models$	not gray	
f1	T2	$\not\models$	gray	
f2	T1	$\not\models$	gray	NEGATIVE
f3	T1	$\not\models$	gray	NEGATIVE
f1	T2	$\not\models$	not gray	NEGATIVE

f2	T2	⊨	not gray	
f2	T1	⊭	gray	
f2	T1	⊭	not gray	
f3	T2	⊭	not gray	NEGATIVE

Let us include the fact f3 in both cases where the discriminant was positive.

f1,f3	T1	⊨	gray	
f1,f3	T2	⊭	not gray	
f1,f3	T2	⊭	gray	
f2,f3	T2	⊨	not gray	
f2,f3	T1	⊨	gray	NEGATIVE
f2,f3	T1	⊭	not gray	

So we can discriminate theories and find T2 as being the most specific one. The critical element in that decision is the fact f3 - *elephant <- royal-e.*

Now, what would happen if the knowledge we have of Clyde and the elephants was imbedded into a larger frame where we assume that *gray* is a *drab* color. Theorist would generate one theory (Fig. 3) and Theory Preference would then be helpless to deny that theory. What we need is a way to enforce that *Clyde is not gray* is a lemma and cannot be later contradicted. We would normally think of including the lemma as a fact but it appears that this would not work properly with the current interpretation of Theorist.

The argument of the next section will attempt to demonstrate that those intermediary truths have the status of lemmas and should be included in the set of facts. We will advocate that the set of facts should be partitioned into extensions. We will argue that the implementation of Theorist should follow strictly the definition given in Poole (1986) and should not use contradictory facts while building theories; morever, only contradictory facts should inhibit the selection of defaults. The current version of Theorist shows some erratic behavior in this respect; sometimes it does not use contradictory facts and sometimes it does. We found that Theorist made use of contradictory facts when it did not have to make use of a default in order to answer a query; it then used facts only. Otherwise it did not generate theories including a default if that default was also a fact. Those choices are in conflict with the interpretation we want to give to the facts. This argument is going to be refined later when we revise the notion of a fact.

A classification of subgraphs

The classes of subgraphs that lead to conclusions according to the axioms of Touretzky are summarized in Figure 4. Classes of subgraph 1, 2, and 3 lead to a unique conclusion while graphs of type 4 lead to two alternative conclusions that cannot be resolved (by using the axioms). Alternative extensions have to be introduced in order to handle alternative conclusions. Graphs of type 2 illustrate the contradiction mechanism of Touretzky while graphs of type 3 illustrate the preclusion

mechanism. Graphs of type 3 have an additional arc over graphs of type 4 so that a possible extension <+a,+d> is precluded by the other, <+a,-d>.

The arcs shown in the figure may themselves be subgrahs but either they end-up being one arc (a conclusion) or two extensions are generated each one including one of two contradictory conclusions. By contrast the current implementation of Theorist builds the set of all conclusions that can be drawn from the original set of facts and defaults by trying all combinations of defaults with the facts.

Defective theories

The idea of a defective theory is that such a theory can never be found in a grounded expansion. Since much on the later discussion is to be based on such grounded expansions, we want to get rid of the defective theories right away. It appears that all theories generated by Theorist which are defective are the ones which are built in the presence of contradictory defaults. For example (Fig. 3) the second theory generated to explain *gray* is defective. It implicitly makes use of a conclusion saying *gray <- royal-e* when there is a given default *n(gray) <- royal-e* which contradicts that conclusion.

The first way around that problem is to duplicate all the theories that may lead to defective theories as facts but it is difficult, beforehand, to know which defaults to pick. The next step is to duplicate all the defaults as facts; which is our preferred approach for the reason that we will later want to include *lemmas* as facts. The problem here appears to be an implementation one; the current version of Theorist does not support the approach; however, we believe it should. The other alternative is to modify Theorist so that it enforces a consistency checking that includes the defaults. This last approach may be the most sensible one but we fixed our choice on the other approach where all defaults are duplicated as facts; this is going to be our work hypothesis for awhile although we will have to abandon it later.

Definition (Defective theory) A theory is said to be defective if it embeds a chain of reasoning stating that some d is true and either the set of facts or the set of defaults includes $\neg$d.

Theorem 1 A defective theory can not be in the conclusion set of a grounded expansion. Proof. By the definition of contradiction given in Touretzky (1984b).

Corolary 1.1 Assuming that Theorist does not generate defective theories, the theories generated by Theorist would not include conclusions based on paths such as $<+a,+b,+c>$ (Fig. 4) if there was a subgraph of type 2 in the same extension.

Corolary 1.2 Since the defective theories can not be in a grounded expansion, any semantics aimed at comparing theories, that may include defective theories, with the idea of merging expansions would be at best suspect.

Why do we need to handle extensions explicitly

Now that we are confident that all theories generated by Theorist could conceivably be in a grounded expansion (extension) we need a framework so that we can associate the intermediary conclusions (lemmas) with each extension since those lemmas will be needed later in order to discriminate between theories. The following theorem establishes under which conditions Theory Preference may

be expected to pick a theory over another..

Theorem 2 Theory Preference can not discriminate between theories which are grounded in different grounded expansions.
Proof. Grounded expansions differ by some conclusions which are true in one expansion and false in another extension. Assume two such grounded expansions induced by the conclusion C where C happens to be true in Extension1 and false in Extension2. If we include all the conclusions in the set of facts (F) for both extensions and create a partition of the conclusions such that P denotes the set of conclusions which are common to both Extension1 and Extension2 and S1 and S2 are sets of conclusions which depend respectively on (P,C) and (P,¬C) we can express the discriminant for Theory Preference as follows:

$(P, C, S1), D1 \models O1$
$(P, C, S2), D2 \not\models O2$
$(P, C, S2), D2 \not\models O1$
and
$(P, \neg C, S2), D2 \models O2$
$(P, \neg C, S1), D1 \not\models O1$
$(P, \neg C, S1), D1 \not\models O2$

where Extension1 supports theory D1 explaining observations O1 and Extension2 supports theory D2 explaining observations O2. One theory could be prefered over the other if, by introducing an additional fact, we could invalidate one discriminant while still satisfying the other one. However, S1 and S2 are sets of conclusions which depend respectively on (P,C) and (P,¬C). Thus a fact that would allow us to discriminate O1 from O2 would have to come from either S1 or S2. S1 and S2 are disjoint sets since in S2 all conclusions imply ¬C while none implies ¬C in S1; instead all conclusions in S1 are supported by C. Thus there will always be a conclusion that can invalidate either discriminant and Theory Preference will not discriminate between two grounded expansions.

Corolary 2 Theory Preference can discriminate between theories that are supported by different extensions if the extensions are not grounded expansions. This is equivalent to say that the extensions are not in a stable state since all the facts have not been accounted for yet. For example, we may see in figure 2 that two extensions coexist as long as the additional fact <+royal.elephant, +elephant> has not been introduced.

The fact C above does not have to be one of the facts or defaults initially given; it may be a derived conclusion. For an example, look at Figure 2 where the facts C and ¬C are <+clyde, +gray.thing> and <+clyde, -gray.thing>; this fact C differs from the one that helps discriminate between the two extensions in Figure 1: namely, <+royal.elephant, +elephant>.

From this discussion and the illustration of how Theory Preference works in our framework it is concluded that Theory Preference should handle correctly the situations of type 3 as illustrated in Figure 4 as long as the pertinent conclusions are available to it. The pertinent conclusions here are any conclusions that may be used in place of the arcs in the type 3 graph. Note that, in the argument, Theory Preference is applied to discriminate between extensions only and that there must be an additional fact or default not already in the extensions that may help prefer one extension over the other. Situations of type 4 are handled through the multiple extension mechanism; Theory Preference is unconclusive in those cases.

Another look at the semantics

We assume here that we have a tree of extensions that has been built by including all facts and defaults except for the ones that may be used to discriminate between extensions (the horizontal arrow in Figure 3, type 3 diagram). Let us call St the set of all facts and defaults that are used to build the tree and Sd the ones that are used only to discriminate between extensions. We may assume that the tree includes only grounded expansions of the set Sd and that it does not include any defective theory. That tree is now going to be pruned by including the discriminating facts and defaults (Sd) - the ones we left aside when building the tree. This is where Theory Preference can be used as a pruning criterion; the tree is a binary tree and Theory Preference is computed to decide if one branch may be cut. The pruned tree shall contain all the correct theories. This is not a proposed algorithm; it illustrates how Theorist and Theory Preference each contribute one aspect of the semantics in the presence of a tree of extensions.

The basic semantic characterization of Theorist is that the generated theories together with the facts be consistent and that they explain the observations. The consistency requirement is the same as the requirement by Touretzky that newly inferred facts do not contradict conclusions. The fact that the current version of Theorist generates defective theories is believed to be implementation dependent. On another hand, Touretzky provides the mechanism of multiple extensions to handle the case where there are contradicting conclusions. The inferential distance concept of Touretzky has no equivalent in Theorist; it is addressed rather by Theory Preference.

The inferential distance concept is a mechanism to relate two extensions. It is still a non local concept if we look at it in the context of the extensions; it is in some way a mechanism for pruning the tree of grounded expansions. Thus we may expect that some of the initial facts and defaults have a role in pruning the tree while others help in generating the tree. This is precisely the role that is being attempted by Theory Preference in the context of Theorist; it is looking for a pruning fact that will allow the Poole's discriminant to select one of two theories. What is not being addressed explicitly in the Poole (1985) paper is that the theories being compared are in different extensions. This is critical since Theory Preference is to make use of lemmas and those lemmas have to be labeled by the extension to which they belong.

What are the facts on which theory preference is based

We have obviously abused of the notion of a fact up to now and we want to correct the situation at least slightly. We have made it a work hypothesis that conclusions (lemmas) derived by Theorist should be included with the facts. In that case the word *fact* is used in the technical denotation it has in Theorist: a set of clauses that are used in some specific way. The usual meaning of a fact however is strictly extensional; it refers to beliefs about a specific state of affairs in the external world. This is to be opposed to the more intentional denotation of the defaults. This interpretation seems to be an intuition behind Theorist.

Following those premises there should not be much ambiguity as to what the facts are and what the defaults are for any given world, and arguments as to when include an information in the facts and when to include it in the defaults should be based on where the information comes from: from an observation or from a deduction. Observed facts which apply only to the world under observation are

included with the set of facts and the rules that may apply to multiple worlds are included as defaults.

We are left with the problem of where to put the lemmas. According to the previous discussion, the lemmas are neither facts nor defaults. Additionally, the lemmas are not needed by Theorist; they are only needed by Theory Preference where they may act as precluding clauses while comparing theories. Additionally, lemmas are consistent in some extensions and inconsistent in other extensions. We conclude that the lemmas shall be included with the theory itself. Thus the work hypothesis we have used so far where the lemmas are included with the facts does not hold anymore. Fortunately the other conclusions that we made are not changed. Now we have the problem of how to mix the defaults and the facts in a theory.

There is a difficulty in implementing the concept of intensional denotation and to mix it with Prolog since Prolog is basically designed to handle extensional concepts. As an example let compare the following two sets of clauses. The first set has an extensional expression and leads to a contradiction.

```
default fly(X);
default n(fly(emu));
```

The second set has an intentional expression and could be handled properly by Theorist if the modification proposed earlier was adopted.

```
default fly <- anything;
default anything <- emu;
default n(fly) <- emu;
fact emu;
```

How is the expression in the first set to be handled is highly implementation dependent. If we adopt the semantics of Prolog, then `fly(X)` and `n(fly(emu))` are both basic defaults (at least they could be if they were derived) and we have to create two theories (extensions) to accomodate them - this is not what we want. We may define another semantics in order to interpret the first set. We may impose that `fly(X)` be a shorthand of something like `fly <- X` and `X <- emu` so that the extensional clause `fly(X)` becomes two intensional clauses. Can we induce from that example that an extensional clause with a variable in it becomes many intensional clauses? In any case we apparently need some way of translating between intensional and extensional clauses since both are to be used together in the context of Theorist.

Conclusion

It is first concluded that Theorist should be modified so that it does not generate defective theories; decisions of type 2 (Fig. 4) would then be handled by Theorist itself. Two arguments that support that conclusion are one to provide a correct treatment of lemmas and second to make sure that the generated extensions are proper extensions and thus can be compared by Theory Preference under a meaningful semantics. An argument against our conclusion is that one wants to keep the defective theories in order to perform diagnostics.

We have seen that the application of the axioms of Touretzky allows us to come up with a set of definite conclusions which are derived from the initial set of defaults. Any definite conclusion may

be used to replace all the defaults that were used to derive it between two nodes in the graph; definite conclusions are lemmas. The lemmas are not necessary to Theorist itself but are needed by Theory Preference to select the most specific theory. It is thus concluded that the lemmas should be added to the generated theory rather than to the facts.

The current concept of Theorist skips the question of what to do with intensional statements. This leads us into an additional difficulty when atoms include variables since Theorist (modified and not modified) would then blindly generate two theories when a clause is syntactically contradictory but should be interpreted, semantically, as a decision of type 2 (Fig. 4). We have proposed a way to look at the problem but the question is basically unsolved.

A Prolog implementation of the axiomatization of Touretzky is currently in development and the above discussion reflects some of the insights gained while doing the implementation.

A result gained from the work is a better understanding of both Theorist and the axiomatization of Touretzky. This gives us some clues on how to make modifications to Theorist. For example, one way to build a system that is consistent with the results obtained by using the method of Touretzky might be to integrate Theory Preference within Theorist itself so that Theorist can build a set of lemmas that can be used in the derivations. The lemmas would then have to be kept in a separate structure instead of within the theories; the lemmas would then be labeled by the set of theories to which they belong. The other alternative is the *generate and test* approach where one first generates the tree of all the possible theories (extensions) by using Theorist (revised) and then prune the tree with Theory Preference (also revised). This last approach, although simpler, is more sensitive to explosions in the number of theories.

References

T. Imielinski (1985) "Results on translating defaults to circumscription", *IJCAI-85*, pp. 117-20.

V. Lifschitz (1985) "Computing circumscription", *IJCAI-85*, pp. 121-7.

J. McCarthy (1985) *Applications of Circumscription to Formalizing Common Sense Reasoning*, Dept. of Computer Sciences, Stanford University, Report No. STAN-CS-85-1077. Also in Artificial Intelligence.

D.L. Poole (1985) "On the Comparison of Theories: Preferring the Most Specific Explanation", *IJCAI-85*, pp. 144-7.

D.L. Poole (1986) *Default Reasoning and Diagnosis as Theory Formation*, Dept. of Computer Science, University of Waterloo, Research Report CS-86-08.

D.L Poole, R. Goebel, and R. Aleliunas (1986) *Theorist: A Logical Reasoning System for Defaults and Diagnosis*, Dept. of Computer Science, University of Waterloo, Research Report CS-86-06.

R. Reiter (1980) "A Logic for Default Reasoning", *Artificial Intelligence*, vol. 13 (1980), pp. 81-112.

D.S. Touretzky (1984a) "Implicit ordering of defaults in inheritance systems", *AAAI-84*, pp. 322-5.

D.S. Touretzky (1984b) *The Mathematics of Inheritance Systems*, PhD Thesis, Computer Sciences Dept., Carnegie-Mellon University, Report CS-84-136.

- o The *implicit* handling of exceptions.
- o Major theorems of the thesis. Γ denotes an inheritance network
  - Theorem 2.5. If $\langle x_1,...x_n\rangle$ is an element of a grounded expansion of Γ, Then x_1 through x_{n-1} are positive.
  - Theorem 2.7 Every grounded expansion of an i.a. Γ is finite. (i.a. - IS-A acyclic)
  - Theorem 2.8 A grounded expansion of Γ is consistent iff Γ is.
  - Theorem 2.11 Every i. a. Γ has a constructible grounded expansion.
  - Theorem 2.13 If a totally acyclic Γ is ambiguous, then each of its grounded expansions is unstable.
  - Theorem 2.14 If an acyclic Γ has an unstable grounded expansion then it has more than one grounded expansion.
- o The inheritance language.
  - * Predicates and individuals.
  - * Tokens, which are signed predicates or individuals.
  - * Inheritance assertions: ordered pairs of tokens.
  - * Inheritance paths: sequences of tokens of length > 1.
  - * Inheritance networks: sets of inheritance assertions.
  - * Inheritance theories (expansions): closed sets of inheritance paths.

  objects of discourse. The symbols representing individuals and predicates.

  real world predicate. The are assumed to be binary.

  token. They are derived from real-world individuals and predicates: positive (+), negative (-), or neutral (#).

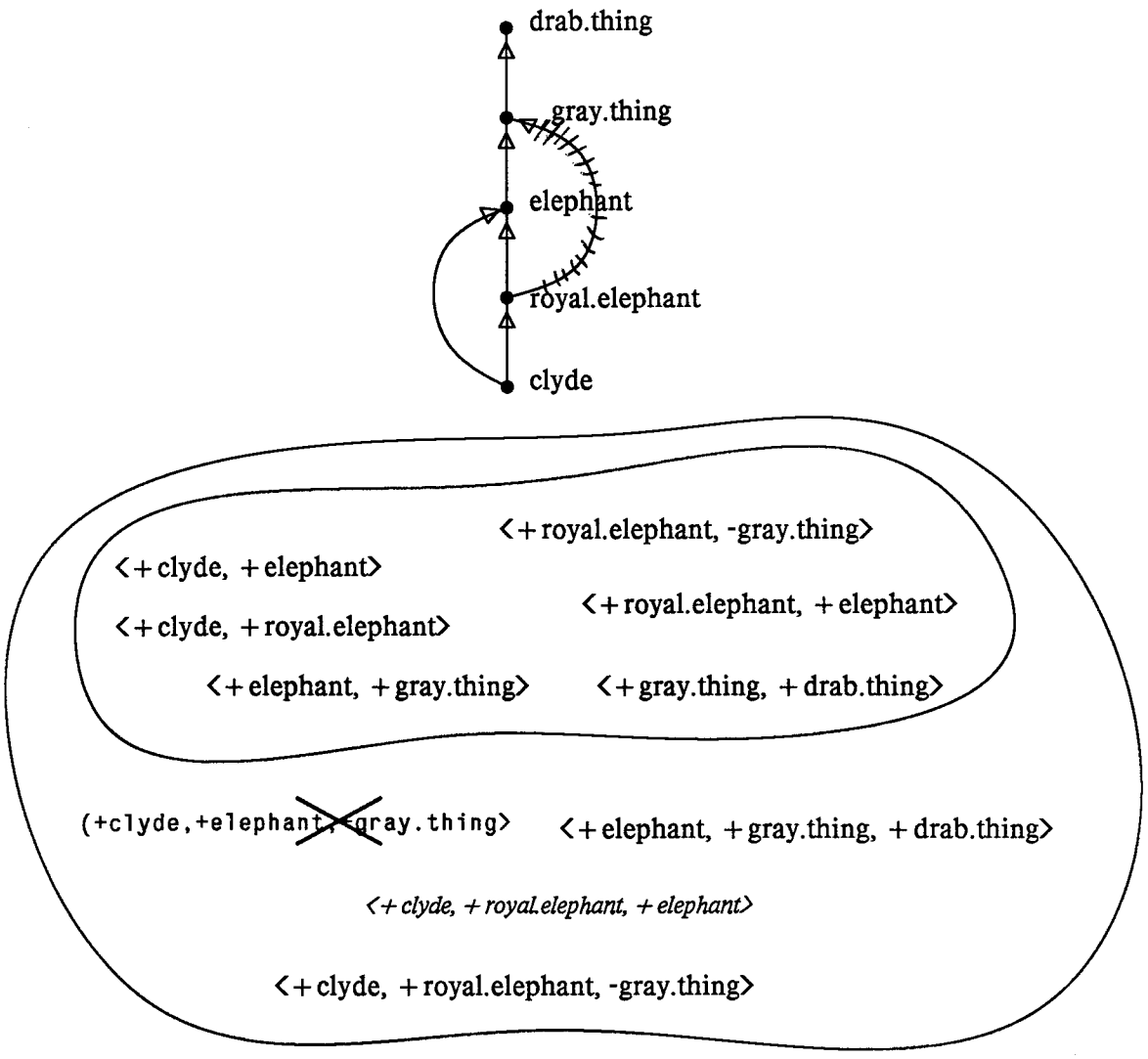
  Π The set of real-world individuals and predicates referred to in an inheritance system.

  Θ The set of all tokens derivable from Π. $\Theta = \text{def}\ \{+,-,\#\} \times \Pi$
  inheritance assertions. They are elements of $\Theta \times \Theta$. There are six pairs of well-formed pairs. Let define: a-individual and p,q-predicates.
  - $\langle +a,+p\rangle$ a is a p;
  - $\langle +a,-p\rangle$ a is not a p;
  - $\langle +a,\#p\rangle$ No conclusion whether a is a p.
  - $\langle +p,+q\rangle$ p's are q's;
  - $\langle +p,-q\rangle$ p's are not q's;
  - $\langle +p,\#q\rangle$ No conclusion whether p's are q's.

  Definition 2.6 Φ is *closed under inheritance* iff Φ contains every sentence inheritable in Φ.

  Definition 2.7 Φ is an *expansion* of a set $S \subseteq \Sigma$ iff $S \subseteq \Phi$ and Φ is closed under inheritance.

  Definition 2.8 Φ is *grounded* is a set $S \subseteq \Sigma$ iff every sequence in Φ - S is inheritable in Φ.



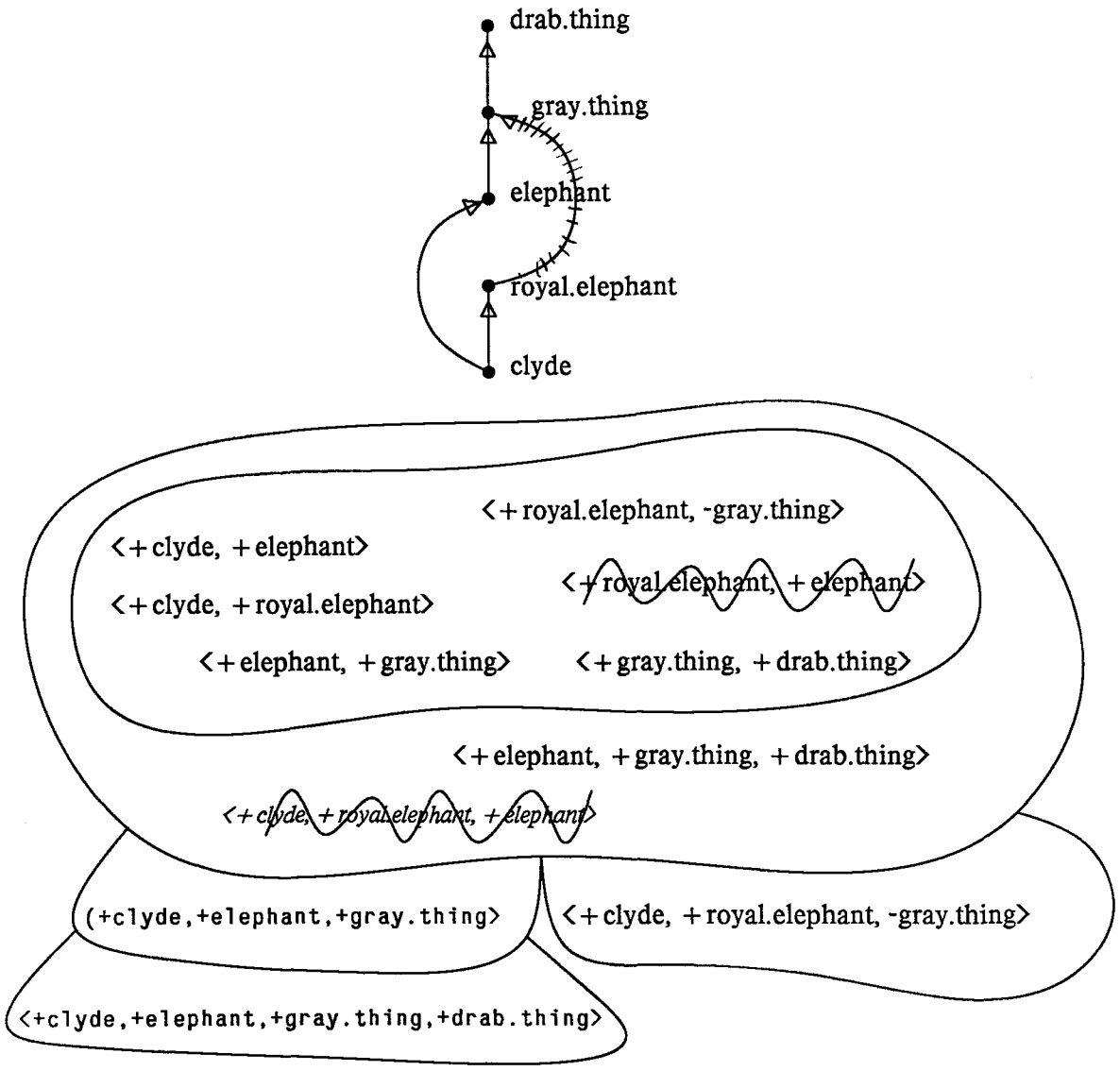
THEORIST

<+ clyde, + elephant> <+ elephant, + gray.thing> <+ gray.thing, + drab.thing>

<+ clyde, + royal.elephant> <+ royal.elephant, + elephant>
<+ elephant, + gray.thing> <+ gray.thing, + drab.thing>

<+ clyde, + royal.elephant> <+ royal.elephant, -gray.thing>

Figure 1. The Clyde example. The grounded expansion (middle) and chains of reasoning generated by Theorist (bottom).



THEORIST

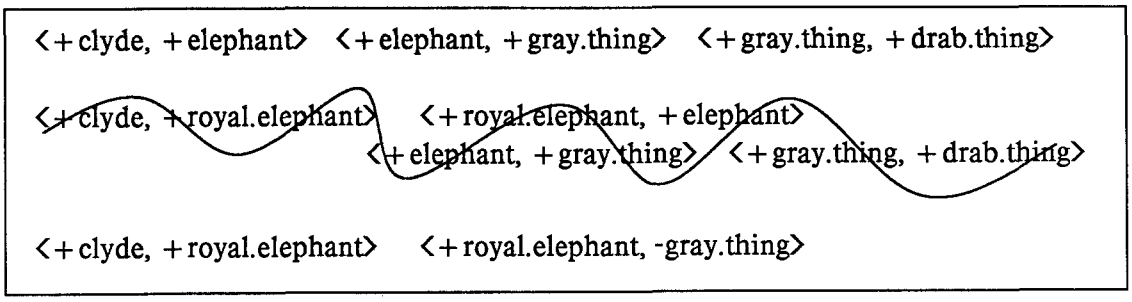


Figure 2. The Clyde example modified to illustrate the role played by the rule <+ royal.elephanr, + elephant>. Two grounded expansions (middle) and the chains of reasoning generated by Theorist (bottom).

```
                default d(1): drab <- gray;
                default d(2): gray <- elephant;
                default d(3): elephant <- clyde;
                default d(4): elephant <- royal-e;
                default d(5): n(gray) <- royal-e;
                default d(6): royal-e <- clyde;
                rule clyde;

: explain drab;
  yes
    answer: [drab]
    theory: drab <- gray;
            gray <- elephant;
            elephant <- clyde;
  yes
    answer: [drab]
    theory: drab <- gray;
            gray <- elephant;
            elephant <- royal-e
            royal-e <- clyde;
  no (more) answers


: explain n(drab);
  no (more) answers

: explain gray;
  yes
    answer: [gray]
    theory: gray <- elephant;
            elephant <- clyde;
  yes
    answer: [gray]
    theory: gray <- elephant;
            elephant <- royal-e;
            royal-e <- clyde;
  no (more) answers

: explain n(gray);
  yes
    answer: [n(gray)]
    theory: n(gray) <- royal-e;
             royal-e <- clyde;
  no (more) answers
```

Figure 3. The elephant problem solved by Theorist: edited Theorist listing.

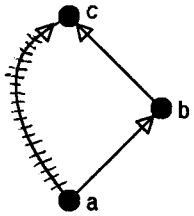
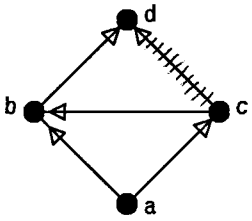
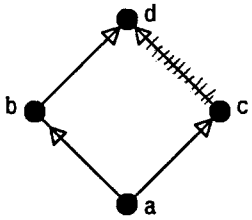
	GRAPH	CONCLUSION
1)		<+a,+d>
2)		<+a,-c>
3)		<+a,-d>
4)		Extension1: <+a,+d> Extension2: <+a,-d>

Figure 4. Classes of subgraph addressed by the axiomatization of Touretzky. The conclusion drawn from the situation illustrated on the left column is shown on the right.

Temporal Explanation: Prediction and Retrodiction

Scott D. Goodwin
Logic Programming and AI Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA N2L 3G1
sdgoodwin@waterloo.CSNET
(519) 885-1211 ext. 6201

Abstract

Recently, much interest has been focused on nonmonotonic reasoning in temporal domains. Since Hanks and McDermott [Hanks86] discovered that temporal axiomatizations give rise to the multiple model/multiple extension problem, many solutions have been proposed. Researchers seem to have reached a consensus that what is needed is some sort of model ordering (or theory preference) scheme [Lifschitz86, Kautz86, Goebel87]. In what follows, nonmonotonic reasoning in temporal domains is considered from the perspective of theory formation [Poole87]. In particular, I try to explain the relationship between the theory formation methodology and the philosophy of science. The notion of explanation is examined and two important types of temporal explanation are defined: *prediction* and *retrodiction*. An important distinction is made between the *task of predicting* some future situation and the *task of (retrodictively) explaining* some past situation in light of what is known in the present. These ideas are used to examine the nature of preferred explanations in temporal domains. It is suggested that preferred temporal theories for prediction tasks are ones that use predictive arguments and that preferred temporal theories for explaining tasks are ones that use retrodictive arguments. The main point of the paper is that discriminating between theories by their temporal structure is in accord with the philosophy of science.

1. Introduction

Recently, much interest has been focused on nonmonotonic reasoning in temporal domains. Since Hanks and McDermott [Hanks86] discovered that temporal axiomatizations give rise to the multiple model/multiple extension problem, many solutions have been proposed. Researchers seem to have reached a consensus that what is needed is some sort of model ordering (or theory preference) scheme [Lifschitz86, Kautz86, Goebel87]. In what follows, nonmonotonic reasoning in temporal domains is considered from the perspective of theory formation [Poole87]. In particular, I try to explain the relationship between the theory formation methodology and the philosophy of science. The notion of explanation is examined and two important types of temporal explanation are defined: *prediction* and *retrodiction*. An important distinction is made between the *task of predicting* some future situation and the *task of (retrodictively) explaining* some past situation in light of what is known in the present. These ideas are used to examine the nature of preferred explanations in temporal domains.

2. Theory Formation

The development of nonmonotonic reasoning systems was, in part, motivated by the inability of logical deduction to capture certain forms of rational inference. In order to draw useful conclusions about a domain for which our knowledge is incomplete or inaccurate, we must make assumptions. Given additional information, we may retract some of our assumptions, and with them, the conclusions they entail. Because logical deduction is monotonic and because rational inference is not, attempts were made to extend classical logic to allow nonmonotonic reasoning [McCarthy80, Reiter80, McDermott80].

Israel has critisized these formalizations of nonmonotonic reasoning and argued instead that nonmonotonic reasoning should be considered within the framework of "scientific" theory formation [Israel80]. Such an approach has been taken by Poole and his colleagues [Poole87] in the Theorist project at the University of Waterloo. Based on a philosophy inspired by Popper [Popper58], Theorist views reasoning as "scientific"[1] theory

[1] Or more precisely, prescientific.

formation (rather than as deduction). Science is concerned, not merely with collecting facts, but with explaining them. Consequently, reasoning in the Theorist framework involves building theories that explain observations.

The scientific method suggests a general methodology for knowledge representation and reasoning in AI. We can represent our domain knowledge as facts, and then provide possible hypotheses to supplement our incomplete domain knowledge. From these possible hypotheses, theories can be constructed to explain our observations. When there are multiple potential explanations, then theory preference can be used to choose between them. Thus, in addition to the facts and possible hypotheses, domain dependent theory preference heuristics (heuristics because theories are potentially refutable) must be defined.

3. Explanation

What is it to give a scientific explanation? The deductive-nomological model of explanation [Hempel48] defines an explanation as a valid deductive argument. The premises of the argument include nomological general statements (laws) and other singular statements (called initial conditions). These statements deductively entail the conclusion. In this sense, the conclusion is explained by the premises, that is, to explain a state of affairs is to deduce its description from a law. Thus, scientific explanation involves a subsumption argument under laws.

A distinction can be made between *potential* explanations and *actual* explanations. A potential explanation is a valid deductive argument whose premises entail the conclusion. For an explanation to be an actual explanation, its singular premises must be true and its general premises must be well-confirmed, lawful generalizations.

In the above, explanation takes the form of a deductive argument. If the explanatory premises are true, a deductive argument provides conclusive evidence for the conclusion. When our premises are not so certain as to allow such conclusive explanations (as is

usually the case), we have at least two alternatives: we can turn to probabilistic explanations (cf. [Rescher70, Hempel65, Carnap50]), or we can tentatively assume the premises are true and treat the explanation as potentially refutable. The latter mode of explanation is described by Popper [Popper58].

In this approach, the premises of a deductive argument can be of two types: those that we accept as true, and those we treat as assumptions. These two types of premises taken together form a potential explanation of the deductively entailed conclusions. In the spirit of the scientific method, a potential explanation can be subjected to crucial experiments, and having survived the tests, the potential explanation becomes well-confirmed. Thus Popper writes:

> "A scientist, whether theorist or experimenter, puts forward statements, and tests them step by step. In the field of the empirical sciences, more particularly, he constructs hypotheses, or systems of theories, and tests them against experience by observation and experiment." [Popper58, p. 27]
>
> "From a new idea, put up tentatively, and not yet justified in any way — an anticipation, a hypothesis, a theoretical system, or what you will — conclusions are drawn by means of logical deduction. These conclusions are the compared with one another and with other relevant statements, so as to find what logical relations (equivalence, derivability, compatibility, incompatibility) exist between them." [Popper58, p. 32]

This is the philosophy underlying the Theorist framework [Poole87]. Instead of viewing reasoning as deduction from our knowledge, reasoning may be better modelled by scientific theory formation. Consequently, reasoning in the Theorist framework involves building theories that explain observations. While the intuition underlying Theorist stems from the deductive nomological model of explanation described above, there is a notable difference. The Theorist framework does not require explanations to contain lawful general statements. Thus, explanations in Theorist are usually only potential explanations. Because of this, it may be more appropriate to think of Theorist's explanations as "prescientific" (or commonsense) theories.

The current implementation of Theorist (all examples in this paper were run on Theorist version 0.21, which compiles statements to Waterloo Unix Prolog)[2] uses full

[2] See [Cheng84]

first-order clausal logic as its representation language. Statements are divided into two types: *facts* and *possible hypotheses* (or defaults). Facts are statements that we accept as true and thus constitute ordinary assertions in a logical theory. Possible hypotheses, however, are *given a different epistemological status* than facts. They are used as a kind of schema for axioms — possible hypotheses can be viewed as the specification for the generation of logical theories which are extensions of the facts. Instances drawn from the set of possible hypotheses can be used to construct consistent explanations for a set of observations (or desired conclusions).

Let F be the set of facts, Δ the set of possible hypotheses, $A=\{d_1, d_{2,...}\}$ the set of instances of elements of Δ, and G the set of observations. Theorist reasons by first trying to deduce G from F. Failing that, it next tries to find elements of A which conjoined to F allow G to be deduced. Thus, Theorist constructs explanations of the form $F \cup T_i$ where T_i is a subset of the elements of A (i.e., a set of instances of possible hypotheses). In addition to the requirement that $F \cup T_i$ constitute an explanation of G, that is, $F \cup T_i \models G$, there is also the requirement that the explanation be consistent. In other words, we require that $F \cup T_i$ be consistent.

This consistency check can be viewed as submitting a potential explanation to "scientific" testing (i.e., comparing the consequences of the theory with what is already known). Note that determining the consistency of $F \cup T$ is, in general, only semi-decidable. As well, determining whether G is a theorem of $F \cup T$ is, in general, only semi-decidable. Therefore, the process of explanation described above is completely undecidable. This should not be seen as blemish on the methodology underlying Theorist; rather, we should be surprised if theory formation were decidable. In practice, we could submit our explanations to a partial consistency check. Passing the partial consistency test can be viewed as confirming evidence for the potentially refutable explanation. This view of reasoning has a solid basis in the philosophy of science (cf. [Popper58, Rescher70,

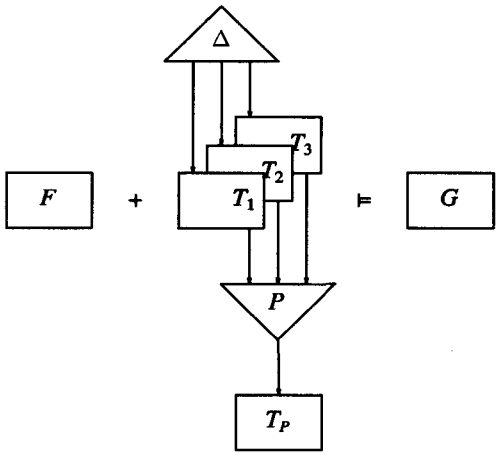
Hempel65]).

In deductive logic, there can be many arguments for the same conclusion; similarly, in theory formation, there can be many potential explanations of the same observation. This corresponds to the multiple model/multiple extension problem (cf. [Reiter80, Hanks86]). Faced with a multiplicity of explanations and a desire to draw useful conclusions, we are often forced to chose between the competing theories.

The issue of theory preference has been considered for several domains. In domains with inheritance hierarchies, the explanation that uses the most specific knowledge is preferred [Poole85]; while in learning domains, the most general explanation is desired [Poole87]; in diagnosis, the most probable, the most serious, or the most specific explanation may be sought [Poole87, Jones85, Poole86]; in analogical reasoning, the simplest and most relevant explanation is preferred [Jackson87]; and in planning, the chronologically most persistent explanation is chosen [Goebel87]. Preference criteria for vision [Gagné86] and for user modelling [Arragon86] are also being investigated. Figure 1 illustrates the theory formation/theory preference framework.

4. Prediction and Retrodiction

When the element of time is crucially involved in an explanation, we call it a *temporal* explanation. Two important kinds of temporal explanation are described by Rescher [Rescher70]: *prediction* and *retrodiction*. A *potential* prediction has the structure of a potential explanation where a time dependent conclusion follows from premises consisting of general statements and time dependent initial conditions. As well, the time parameters of the premises must predate the time parameter of the conclusion. For example, the following is a potential prediction of $Q(t)$:



Find T_i, instances of elements of Δ, such that:
$F \cup T_i \models G$, and
$F \cup T_i$ is consistent
where Δ is a set of defaults
T_i is a set of instances of defaults
F is a set of facts
G is a goal (or its negation) to be explained.
Then choose $T_P \in \{T_1, T_2, \ldots\}$
such that $\forall i \; T_P >^P T_i$
where $>^P$ is a theory preference heuristic
T_P is a preferred theory
P is a procedure for choosing a preferred theory.

Figure 1. Theory Formation/Theory Preference Framework

$$\frac{C_1(t_1), \ldots, C_m(t_m) \qquad \text{(Initial Conditions)}}{\dfrac{L_1, \ldots, L_n}{Q(t)} \qquad \text{(universal generalizations)}} \qquad \text{(conclusion)}$$

provided each $t_i < t$. Here each premise C_i is a time dependent initial condition and each premise L_i is a universal generalization. The time dependent conclusion Q is a logical consequence of the premises. In addition to the requirements of a potential prediction, an *actual* prediction must have fact-asserting singular premises, and well-confirmed, law-asserting general premises. Furthermore, the time parameters in the predictive argument are constrained as follows: if the present time is t_{now} then each $t_i \leq t_{now} < t$.[3]

[3] See [Rescher70, pp. 33ff.] for a discussion of the need to constrain the chronological direction of predictive

A second form of temporal explanation is *retrodiction*. A *potential* retrodiction has the structure of a potential explanation where a time dependent conclusion follows from premises consisting of general statements and time dependent initial conditions. As well, the time parameters of the premises must postdate the time parameter of the conclusion. For example, the following is a potential retrodiction of $Q(t)$:

$$\begin{array}{ll} C_1(t_1), \ldots, C_m(t_m) & \text{(Initial Conditions)} \\ \underline{L_1, \ldots, L_n} & \text{(universal generalizations)} \\ Q(t) & \text{(conclusion)} \end{array}$$

provided each $t_i > t$. In addition to the requirements of a potential retrodiction, an *actual* retrodiction must have fact-asserting singular premises, and well-confirmed, law-asserting general premises. Furthermore, the time parameters in the retrodictive argument are constrained as follows: if the present time is t_{now} then $t_{now} > t_i$.

As with ordinary explanation, we can transform the definition of scientific prediction and retrodiction to prescientific (or commonsense) prediction and retrodiction by dropping the requirement that the general premises be lawful.

5. Temporal Explanation and the Frame Problem

The distinction between prediction and retrodiction can shed light on recent work on the frame problem. To their dismay, Hanks and McDermott [Hanks86] discovered that there can be multiple temporal explanations of a goal stemming from the axiomatization of a temporal domain, and that only one of these corresponds to the intended interpretation. Several solutions to this problem have been proposed (cf. [Lifschitz86, Kautz86, Shoham86, Goebel87]). Each solution recognizes that the alternative explanations can be distinguished on the basis of the temporal order of the underlying assumptions.[4] Shoham

arguments.

[4] Theory formation terminology is used in comparing the various approaches as advocated in [Goebel87].

has called the temporal order that corresponds to the intended interpretation the chronological order [Shoham86]. In the theory formation/theory preference framework, the desired explanation can be selected using the theory preference heuristic *chronological maximal persistence* [Goebel87]. Essentially, this preference heuristic involves selecting the preferred theory based on a pre-specified partial ordering of *persistence sets*. A persistence set represents the relations that persist between situations according to the given theory. Intuitively, the partial ordering of persistence sets reflects the relative likelihood of various combinations of relations persisting. If all that is known about this likelihood is that relations are more likely to persist than not, then the partial ordering is determined by set inclusion. This is the intuition behind Kautz's model ordering [Kautz86].

Some insight can be gained by comparing the temporal structure of chronologically maximally persistent (CMP) theories with alternative theories.

	The set of facts:
	Initial Situation:
fact $\neg loaded(0)$;	The gun is not loaded
fact $alive(0)$;	Fred is alive
	Action: load
fact $loaded(do(load,S))$;	The gun is loaded after the action load
	Action: wait (no known changes)
	Action: shoot
fact $\neg alive(do(shoot,S)) \leftarrow loaded(S)$;	Fred dies when shot with a loaded gun
fact $\neg loaded(do(shoot,S))$;	After shooting, the gun is not loaded
	The set of Frame Defaults:
default $loaded(do(A,S)) \leftrightarrow loaded(S)$;	
default $alive(do(A,S)) \leftrightarrow alive(S)$;	

Figure 2. Yale Shooting Scenario

In the Yale Shooting Scenario (Fig. 2), consider whether Fred will be alive after the actions *load*, *wait*, and then *shoot*. Two of the theories which describe the invariance of relations for this sequence of actions are

$$T_1 = \{loaded(do(wait,do(load,0))) \leftrightarrow loaded(do(load,0)),$$
$$alive(do(load,0)) \leftrightarrow alive(0),$$
$$alive(do(wait,do(load,0))) \leftrightarrow alive(do(load,0))\} \text{ and}$$

$$T_2 = \{alive(do(load,0)) \leftrightarrow alive(0),$$
$$alive(do(wait,do(load,0))) \leftrightarrow alive(do(load,0)),$$
$$alive(do(shoot,do(wait,do(load,0)))) \leftrightarrow alive(do(wait,do(load,0)))\}.$$

The statements in the theories are instances of frame defaults that record two different sets of assumptions about the action sequence. Note that these two theories conflict since

$$F \cup T_1 \models \neg alive(do(shoot,do(wait,do(load,0)))) \text{ while}$$
$$F \cup T_2 \models alive(do(shoot,do(wait,do(load,0)))).$$

In our intended model (i.e., the one that corresponds to our intuitions), however, $\neg alive(do(shoot,do(wait,do(load,0))))$ is true (Fig. 3).

Situations:
$1 \equiv do(load,0)$
$2 \equiv do(wait,do(load,0))$
$3 \equiv do(shoot,do(wait,do(load,0)))$

Intended Model Features				
	0	1	2	3
loaded	F	T	T	F
alive	T	T	T	F

T_1 Model Features				
	0	1	2	3
loaded	F	T	T	F
alive	T	T	T	F

T_2 Model Features				
	0	1	2	3
loaded	F	T	F	F
alive	T	T	T	T

Figure 3. Essential Features of Models for the Shooting Example

We have shown in [Goebel87] that T_1 is the CMP theory.

Table 1 illustrates that the form of a CMP theory is such that the properties of each situation on the path described by the CMP theory are supported by purely predictive arguments.

Situations:
$1 \equiv do(load,0)$
$2 \equiv do(wait,do(load,0))$
$3 \equiv do(shoot,do(wait,do(load,0)))$

Property	Explanation
	Theory T_1
$loaded(1)$	$loaded(1)$
$alive(1)$	$alive(1) \leftrightarrow alive(0)$
$loaded(2)$	$loaded(2) \leftrightarrow loaded(1)$, $loaded(1)$
$alive(2)$	$alive(2) \leftrightarrow alive(1)$, $alive(1) \leftrightarrow alive(0)$, $alive(0)$
$\neg loaded(3)$	$\neg loaded(3)$
$\neg alive(3)$	$\neg alive(3) \leftrightarrow loaded(2)$, $loaded(2) \leftrightarrow loaded(1)$, $loaded(1)$
	Theory T_2
$loaded(1)$	$loaded(1)$
$alive(1)$	$alive(1) \leftrightarrow alive(0)$
$\neg loaded(2)$	$\neg alive(3) \leftrightarrow loaded(3)$, $alive(3) \leftrightarrow alive(2)$, $alive(2) \leftrightarrow alive(1)$, $alive(1) \leftrightarrow alive(0)$, $alive(0)$
$alive(2)$	$alive(2) \leftrightarrow alive(1)$, $alive(1) \leftrightarrow alive(0)$, $alive(0)$
$\neg loaded(3)$	$\neg loaded(3)$
$alive(3)$	$alive(3) \leftrightarrow alive(2)$, $alive(2) \leftrightarrow alive(1)$, $alive(1) \leftrightarrow alive(0)$, $alive(0)$

Table 1. Predictive Explanations

For instance, under T_1 every property of each situation is justified by a subset of T_1 (say T_1') such that $F \cup T_1'$ constitutes a predictive explanation. For example, the property $loaded(do(wait,do(load,0)))$ is derived from

$$T_1' = \{loaded(do(wait,do(load,0))) \leftrightarrow loaded(do(load,0))\}$$

together with the fact $loaded(do(load,S))$. None of the premises of the above argument involve situations that postdate the situation of the conclusion, i.e., $do(wait,do(load,0))$.

The same is not true for T_2. In this case, the property

$$\neg loaded(do(wait,do(load,0)))$$

is derived from T_2 together with the facts

$$alive(0) \text{ and}$$
$$\neg alive(do(shoot,do(wait,do(load,0)))).$$

Note that the argument involves the situation $do(shoot,do(wait,do(load,0)))$ which postdates $do(wait,do(load,0))$; that is, $\neg loaded(do(wait,do(load,0)))$ is not explained by a purely predictive argument.

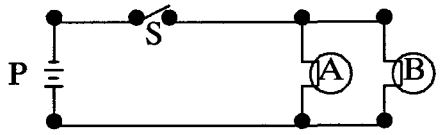
This suggests that when attempting to *predict* the outcome of a course of events, as in the Yale Shooting Scenario, our explanations should be *predictive explanations*. The process of chronological maximization (or minimization) as seen in [Kautz86, Lifschitz86, Shoham86, Goebel87] can be understood as methods of selecting predictive explanations.

Additional insight can be gained by considering the Lighting Circuit Example (Fig. 4). Suppose we know that each of the devices (S,A,B) have the same low failure rate (say .01%). The expected state of affairs at time 2 is characterized by the following theory:

$$T_1 = \{ok(s,1) \leftrightarrow ok(s,0),\ ok(a,1) \leftrightarrow ok(a,0),\ ok(b,1) \leftrightarrow ok(b,0)\}.$$

T_1 can be shown to be the CMP theory for this problem (cf. [Goebel87]). Note that it predictively explains $ok(s,1)$, $ok(a,1)$, $ok(b,1)$, $on(a,2)$, and $on(b,2)$. So here too, the temporal structure of the preferred theory is that of a predictive explanation.

Retrodiction is another important class of temporal explanation. When we are interested in *explaining* some *past* situation in light of what we know in the present situation, *retrodictive explanation* is appropriate. For instance, in the Yale Shooting Scenario, if in the situation after shooting we observe that Fred is alive then we can retrodictively explain that the gun was not loaded in the situation after waiting:



fact $on(s,0)$;

fact $ok(s,0)$;
fact $ok(a,0)$;
fact $ok(b,0)$;

fact $on(s,1)$;

fact $on(a,T+1) \leftrightarrow on(s,T) \wedge ok(s,T) \wedge ok(a,T)$;
fact $on(b,T+1) \leftrightarrow on(s,T) \wedge ok(s,T) \wedge ok(b,T)$;

default $ok(X,T+1) \leftrightarrow ok(X,T)$;

Figure 4. Lighting Circuit Example

$$\frac{alive(do(shoot,do(wait,do(load,0)))) \quad \neg alive(do(shoot,do(wait,do(load,0)))) \leftarrow loaded(do(wait,do(load,0)))}{\neg loaded(do(wait,do(load,0)))}$$

Or in the Light Circuit Example, if at time 2 we observe $\neg on(a,2)$ and $\neg on(b,2)$ then we can use a retrodictive argument to determine the source of the failure. Three of the potential explanations for this situation are

$T_2 = \{\}$;
$T_3 = \{ok(s,1) \leftrightarrow ok(s,0)\}$;
$T_4 = \{ok(a,1) \leftrightarrow ok(a,0),\ ok(b,1) \leftrightarrow ok(b,0)\}$.

Note that T_2, T_3, and T_4 retrodictively explain $\neg ok(s,1) \vee [\neg ok(a,1) \wedge \neg ok(b,1)]$; $\neg ok(a,1) \wedge \neg ok(b,1)$; $\neg ok(s,1)$ respectively. As with predictive explanation, preference can be used to discriminate between competing retrodictive explanations. Of the three theories above, T_4 is the most likely given the failure rate information. In circumscription approaches (cf. [Lifschitz86, Kautz86]), the models corresponding to T_3 and T_4 are minimal (and thus incomparable). In the theory formation framework, it is a simple

matter to include the failure rate information in the theory preference heuristic and thereby select T_4 (cf. [Goebel87]).

6. Conclusion

In the proceeding sections, I have tried to give some philosophical basis to some of the recent work in nonmonotonic reasoning. This work was viewed from the perspective of theory formation. As Goebel and I have proposed elsewhere [Goebel87], it seems that all current forms of nonmonotonic reasoning can be usefully examined in the theory formation framework. The chief aim of the paper was to discuss nonmonotonic reasoning in temporal domains using insights from the philosophy of science. The relationship between theory formation and the philosophy of science led to the idea that it is useful to considered the temporal structure of explanations. An important distinction is made between the task of predicting some future situation and the task of (retrodictively) explaining some past situation. It was suggested that preferred temporal theories for prediction tasks are ones that use predictive arguments. Likewise preferred temporal theories for explaining tasks are ones that use retrodictive arguments. Thus, the main point of the paper is that discriminating between theories by their temporal structure is in accord with the philosophy of science.

Acknowledgements

The Philosophical Engineering and Trendy Formalisms division of the Computer Science department's Logic Programming and AI group has helped mould these ideas. Deep thanks to Randy Goebel and David Poole for their constant encouragement and guidance. Special thanks to Denis Gagné for many useful discussions and helpful criticisms. Thanks to four anonymous reviewers who convinced me to abandon an early draft of a paper quite different from this one.

References

[Arragon86] P. van Arragon (1986), Using Scientific Theory Formation for User Modelling, Department of Computer Science, University of Waterloo, October, 30 pages.

[Carnap50] R. Carnap (1950), *Logical Foundations of Probability*, University of Chicago Press, Chicago, Illinois.

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

[Gagné86] D. Gagné, R. Goebel, and D. Poole (1986), Theory Formation for High Level Scene Analysis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, December, 14 pages.

[Goebel87] R.G. Goebel and S.D. Goodwin (1987), Applying theory formation to the planning problem, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, April 13-15, University of Kansas, Lawrence, Kansas, 207-232.

[Hanks86] S. Hanks and D. McDermott (1986), Default Reasoning, Nonmonotonic Logic, and the Frame Problem, *Proceedings of the National Conference for Artificial Intelligence (AAAI-86)* **1**, August 11-15, University of Pennsylvania,

Philadelphia, Pennsylvania, 328-333.

[Hempel48]
C. Hempel and P. Oppenheim (1948), Studies in the Logic of Explanation, *Philosophy of Science* **15**.

[Hempel65]
C.G. Hempel (1965), *Aspects of scientific explanation and other essays in the philosophy of science*, The Free Press, New York.

[Israel80]
D.J. Israel (1980), What's wrong with non-monotonic logic, *Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80)*, August 18-21, Stanford University, Stanford, California, 99-101.

[Jackson87]
W.K. Jackson (1987), A Theory Formation Framework for Learning by Analogy, Master's dissertation, Department of Computer Science, University of Waterloo, Waterloo, Ontario, January.

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

[Kautz86]
H. Kautz (1986), The logic of persistence, *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, August 11-15, University of Pennsylvania, Philadelphia, Pennsylvania, 401-405.

[Lifschitz86]
V. Lifschitz (1986), Pointwise circumscription: preliminary report, *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, August 11-15, University of Pennsylvania, Philadelphia, Pennsylvania, 406-410.

[McCarthy80]
J. McCarthy (1980), Circumscription—a form of non-monotonic reasoning, *Artificial Intelligence* **13**(1&2), April, North-Holland, Amsterdam, 27-39.

[McDermott80]
D. McDermott and J. Doyle (1980), Non-Monotonic Logic I, *Artificial Intelligence* **13**(1&2), April, North-Holland, Amsterdam, 41-72.

[Poole85]
D.L. Poole (1985), On The Comparison of Theories: Preferring the Most Specific Explanation, *Proceeding of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, August 16-18, UCLA, Los Angeles, Califor-

nia, 144-147.

[Poole86]
D.L. Poole (1986), Default Reasoning and Diagnosis as Theory Formation, CS-86-08, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, March, 19 pages.

[Poole87]
D.L. Poole, R.G. Goebel, and R. Aleliunas (1987), Theorist: a logical reasoning system for defaults and diagnosis, *The Knowledge Frontier: Essays in the Representation of Knowledge*, N.J. Cercone and G. McCalla (eds.), Springer-Verlag, New York, 331-352 [in press].

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

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

[Rescher70]
N. Rescher (1970), *Scientific Explanation*, Collier-MacMillian Canada, Toronto.

[Shoham86]
Y. Shoham (1986), Chronological Ignorance: time, nonmonotonicity, necessity, and causal theories, *Proceedings of the National Conference for Artificial Intelligence (AAAI-86)* **1**, August 11-15, University of Pennsylvania, Philidelphia, Pennsylvania, 389-393.

Reasoning about Simultaneous Actions within the Theorist Framework

CS786 Term Paper
André Trudel

April 20, 1987

Abstract

A multi-agent planner must deal with the following questions:

- Can n actions ($n \geq 1$) occur simultaneously? and if so,
- What is the resulting state?

This paper will show how Theorist can be used to answer the above questions. The approach used is based on the ideas found in [Georgeff 86] which have been extended to use defaults.

1 Introduction

The world is assumed to remain in a particular state until one or more actions occur. For example, if the world is in state s and actions $a_1, \ldots, a_n$ ($n \geq 1$) occur, then the world would be transformed to a new state $s' = succ(s)$. The actions $a_1, \ldots, a_n$ occur simultaneously. The case where $a_1, \ldots, a_n$ occur arbitrarily will not be considered. An example is a_1 followed by a_2 and, a_3 occuring simultaneously with a_1 and a_2 (a_3 starts at the same time as a_1 and finishes at the same time as a_2).

Given a specific state s, this paper will show how defaults can be used to answer the following questions:

- Can actions $a_1, \ldots, a_n$ ($n \geq 1$) occur simultaneously in state s?
- What is true in $succ(s)$?

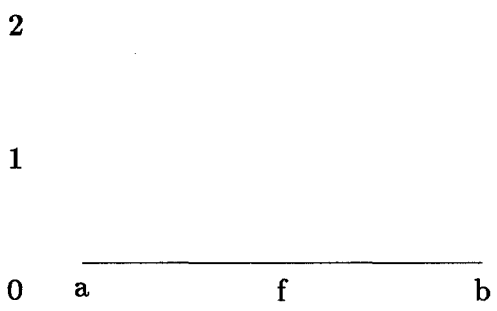


Figure 1: Seesaw example

2 Theorist

Within the Theorist framework [Poole 87], there are two sets of wffs; a set of facts F, and a set of defaults $\triangle$. A wff g is explainable if there is a finite set of ground elements D of $\triangle$ such that:

$$F \cup D \models g \quad \textit{and}$$
$$F \cup D \quad \textit{is consistent}$$

3 A specific example

The seesaw example[1] shown in figure 1 will be used throughout the paper. The world consists of a seesaw with fulcrum f and ends a, b. The possible locations for a, f, and b are 0, 1, and 2. A state is acceptable if a, f, and b are colinear. Initially (named state s_0), a, f, and b are at location 0 (as shown in figure 1).

The seesaw is described using the $loc(X, Y, Z)$ relation which specifies that X is at location Y in state Z. For example, in the initial state, we have:

$$loc(a, 0, s_0)$$
$$loc(f, 0, s_0)$$
$$loc(b, 0, s_0)$$

The action $move(X, Y)$ moves X to location Y. $occurs(move(X, Y), S)$ is used to specify that action $move(X, Y)$ has occured in state S. For example, if we have $occurs(move(a, 2), s_0)$

[1] Example taken from [Georgeff 86].

then $loc(a, 2, succ(s_0))$ will be true.

(2) You mean the direct effects must be consistent with all constraints.

4 Direct effects

When the action $move(X, Y)$ is performed, X will move to location Y. Georgeff [Georgeff 86] calls this the direct effect of the action, and can be axiomatised as:

$$\forall X, Y, S \quad occurs(move(X, Y), S) \rightarrow loc(X, Y, succ(S)) \tag{1}$$

The direct effects can be used to determine if actions can occur simultaneously. Assume we have two actions and each action has one direct effect. If the direct effects have different predicate symbols then in general, the actions can occur simultaneously. For example, if action a_1 has direct effect $p(X)$ and action a_2 has direct effect $q(X)$ then both actions can proceed in parallel. Also, if the direct effects unify then in general, the actions can occur simultaneously. This corresponds to doing the same thing two different ways. For example, if two agents are on either side of a door, one pushing and one pulling, the direct effect of both actions is that the door will be closed. These actions can occur simultaneously. The final case to consider is direct effects that have the same predicate symbol but do not unify. For this case, additional domain knowledge is required. For example, $move(a, 2)$ and $move(f, 1)$ (direct effects $loc(a, 2, succ(s_0))$ and $loc(f, 1, succ(s_0))$) can proceed in parallel but $move(a, 2)$ and $move(f, 0)$ (direct effects $loc(a, 2, succ(s_0))$ and $loc(f, 0, succ(s_0))$) cannot. The additional domain knowledge required is that colinearity must be maintained.

5 Causality

When performed in isolation, an action may affect other relational tuples besides its direct effects. For example, assume that when performed in isolation, the action $move(f, 1)$ will also move a and b to location 1. $move(f, 1)$ can be axiomatised as:

$$\forall S, X \quad occurs(move(f, 1), S) \rightarrow loc(X, 1, succ(S)) \tag{2}$$

There is now a problem, another action such as $move(a, 2)$ cannot be performed simultaneously with $move(f, 1)$. From the direct effect (1) we can conclude that $loc(a, 2, succ(s_0))$ and, from (2) we can conclude that $loc(a, 1, succ(s_0))$.

To solve the problem, Georgeff [Georgeff 86] replaces (2) with a causal law which states that:

> If $move(f, 1)$ occurs and no other action occurs simultaneously which affects the location of a, then $move(a, 1)$ is caused to occur simultaneously with $move(f, 1)$.

The above causal law can easily be implemented using Theorist. First, we add to F axioms which state that at most one action can occur for each of a, f, and b. Secondly, the following default is added to $\triangle$:

$$occurs(move(f,1),S) \rightarrow occurs(move(X,1),S) \tag{3}$$

Whenever default (3) is used, Theorist will perform a consistency check. This check will guarantee the default is used only if no other action affects the location of a.

Causal laws are a very straightforward way of specifying what happens when only a subset of all possible actions occur. But, "causal laws can be quite complex, and may depend on whether or not other actions occur as well as on conditions that hold in the world" [Georgeff 86]. Complex causal laws can result in multiple extensions. For example, assume that when performed in isolation, action $move'(f,1)$ moves f to 1 and a to 2[2]. The causal law for $move'(f,1)$ would be:

$$occurs(move'(f,1),S) \rightarrow occurs(move(a,2),S) \tag{4}$$

If $move(f,1)$ and $move'(f,1)$ occur simultaneously, then using default (3), a is at location 1 and using (4), a is at location 2. We cannot prefer one theory over the other. One of the two *move* actions for a will occur, which one is unknown. A conditional must be included in the plan which checks the location of a after $move(f,1)$ and $move'(f,1)$ have been performed.

6 The Frame Problem

If no action affects the location of a, then we expect a to remain at its current location. This is called the frame problem. The axioms in the previous section cannot be used to solve the frame problem. The frame problem is dealt with by adding the following frame default to $\triangle$:

$$loc(X,Y,S) \rightarrow loc(X,Y,succ(S)) \tag{5}$$

7 Using the axiomatisation

This section will present the complete axiomatisation for figure 1. Examples which use the axiomatisation will then be given.

[2] $move'(f,1)$ and $move(f,1)$ are examples of two distinct actions with identical direct effects.

7.1 The facts F

Direct effects[3]:

$$loc(X, Y, succ(S)) \leftarrow occurs(m(X, Y), S)$$

The initial state s_0:

$$loc(a, 0, s_0)$$
$$loc(f, 0, s_0)$$
$$loc(b, 0, s_0)$$

A state S is valid if the locations of a, f, and b are colinear:

$$validState(LA, LF, LB, S) \leftarrow$$
$$loc(a, LA, S)$$
$$loc(f, LF, S)$$
$$loc(b, LB, S)$$
$$colinear(LA, LF, LB)$$

Definition of colinear:

$$colinear(X, X, X)$$
$$colinear(0, 1, 2)$$
$$colinear(2, 1, 0)$$

Axioms which specify the following are also required:

- at most one move action for each of a, f, and b
- a, f, and b can each be in only one location in a state.

7.2 The defaults $\triangle$

The causal laws:
$m(f, X)$ will cause a and b to move to X. $m(a, X)$ is similar to $m(f, X)$. The exception is $m(a, 2)$ which moves f to 1 and b to 0. $m(b, X)$ is similar to $m(f, X)$.

$$\begin{array}{lcll} occurs(m(Y, X), S) & \leftarrow & occurs(m(f, X), S) & (d1) \\ occurs(m(Y, X), S) & \leftarrow & occurs(m(a, X), S) & \\ & & ne(X, 2) & (d2) \\ occurs(m(f, 1), S) & \leftarrow & occurs(m(a, 2), S) & (d3) \\ occurs(m(b, 0), S) & \leftarrow & occurs(m(a, 2), S) & (d4) \\ occurs(m(Y, X), S) & \leftarrow & occurs(m(b, X), S) & (d5) \end{array}$$

The frame default:

$$loc(X, Y, succ(S)) \leftarrow loc(X, Y, S) \qquad (f1)$$

[3] $move \equiv m$

7.3 Examples

In each of the following examples, the actions must be added to F. For example, in the first example, $occurs(m(a,2),s_0)$ and $occurs(m(f,2),s_0)$ must be added to F. Next, we ask Theorist to explain $validState(X,Y,Z,succ(s_0))$.

7.3.1 Can $m(a,2)$ and $m(f,2)$ occur simultaneously?

The answer would be:

$$\begin{array}{l} yes \\ X = Y = Z = 2 \\ default: \quad d1 \end{array}$$

Note that when determining the location of b, there were two possible theories: $d1$ and $d4$. Theory $d4$ was not considered because of the domain dependent restriction on colinearity.

7.3.2 Can $m(a,2)$ and $m(f,1)$ occur simultaneously?

Theorist would this time respond with two answers:

$$\begin{array}{l} yes \\ X = 2, Y = 1, Z = 0 \\ default: \quad d4 \end{array}$$

$$\begin{array}{l} yes \\ X = 2, Y = 1, Z = 0 \\ default: \quad f1 \end{array}$$

The first answer says that $m(a,2)$ caused $m(b,0)$ to occur. The second answer says that nothing affected the location of b so it remained at the same location. There is no reason to prefer one answer over the other unless we had additional information such as the action $m(b,0)$ is very expensive.

7.3.3 Can $m(a,2)$ and $m(b,1)$ occur simultaneously?

The answer would be *no* because colinearity cannot be maintained.

7.3.4 What happens if $m(f,1)$ occurs by itself?

The answer would be:

$$\begin{array}{l} yes \\ X = Y = Z = 1 \\ defaults: \quad d1, d1 \end{array}$$

8 Conclusions

This paper has shown how Theorist can be used to reason about simultaneous actions occuring in simple domains. The axioms required were direct effects, causal laws and a frame default.

Causal laws are an elegant solution to specifying what happens when an action is performed in isolation. They are also implemented quite naturally using defaults. I plan to investigate the use of causal laws for more complex domains.

9 Acknowledgements

I wish to thank Denis Gagne for comments made on the paper. I also wish to thank Sam.

10 References

[**Georgeff 86**] M. Georgeff (1986), *The Representation of Events in Multiagent Domains.* AAAI-86, Philadelphia, USA, 70-75.

[**Kowalski 79**] R. Kowalski (1979), *Logic for problem solving.* Elsevier North Holland, New York.

[**Poole 87**] D. Poole, R. Goebel, and R. Aleliunas (1987), *Theorist: a logical reasoning system for defaults and diagnosis.* The Knowledge Frontier: Essays in the Representation of Knowledge, N. Cercone and G. McCalla (eds.), Springer-Verlag, New York, 331-352 [in press].

Towards solving the multiple extension problem: combining defaults and probabilities*

Eric Neufeld and David Poole
Logic Programming and Artificial Intelligence Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

May 19, 1987

Abstract

The *multiple extension* problem frequently arises in both diagnostic and default reasoning. That is, in many settings it is possible to use any of a number of defaults or hypotheses. If there is no *semantic* way to choose among them, we may want to prefer one because it is more likely or has some other measureable property: cost, severity, fairness.

We combine probabilities and defaults in a simple unified framework that retains the logical semantics of defaults and diagnosis as theory formation from a fixed set of possible hypotheses. We view probability as a *property* of a theory that can be computed from a what is known and what is hypothesized by a valuation function. We present an algorithm that performs an iterative deepening branch-and-bound search for with the property that the first path found is the most likely. The algorithm does not consider unlikely paths until more likely ones have been eliminated.

We outline a way in which probabilities are not constrained by *a priori* independence assumptions; rather, these statistical assumptions are set up as defaults.

While we use probability as a way of preferring one answer to another, the results apply to any property of a theory having a valuation function meeting some usefulness criteria.

1 Introduction

In many situations, we want to use generalised knowledge, prototypical knowledge or knowledge not always true in our domain.

Examples include:

- default reasoning, where we use an instance of a default in order to predict some proposition, if we can do so consistently [25,20].
- diagnosis, where we hypothesize problems with a system that explain observed malfunctions and are consistent with what we know about the system [24,26,5].

*Draft; to appear in proceedings of third AAAI Uncertainty Workshop.

- natural language understanding, where given some discourse we want to compute the underlying presuppositions to allow for the most meaningful reply [15,18].
- planning, where we hypothesize a sequence of actions that imply completion of a goal[10].
- user modelling, where we hypothesize about the knowledge and beliefs of a user[28].

Here, we consider an alternate use of logic to just finding what deductively follows from our knowledge. We use the idea of *theory formation from a fixed set of possible hypotheses* supplied by the user. From instances of these possible hypotheses we try to build an explanation of our goal that is consistent with all our knowledge. If the possible hypotheses are defaults and the goal is something we are trying to predict, then this characterizes a natural form of default reasoning [22]. If the possible hypotheses are possible diseases or malfunctions of some system and the goal is the observed behaviour of the system then we have model-based diagnosis. Similarly, recognition can be seen as theory formation where the possible hypotheses are the prototypes of the things we are trying to recognize.

In essence we are proposing something like scientific theory formation, but without the problem of generating the hypotheses. (Maybe a better name is construction of explanations.) We assume that the user provides the forms of the hypotheses she is prepared to accept as part of an explanation.

Unless observations together with facts contain complete information about the system, many theories may explain the observations: the "multiple extension" problem [6,12]. In such a framework, it is natural that there may be multiple explanations for why some system is behaving in some way, or why some proposition is expected to be true.

To choose among theories, we must define *theory comparators* Theory comparators seem to fall into two broad classes. The first broad class uses information implicit in the problem statement to choose among consistent theories for one that is correct in the intended interpretation. For example:

- Poole proposes the notion of *most specific theory* [21] to give semantics to the idea that when we have generalised knowledge and more specific knowledge represented as defaults, then we prefer to use the more specific knowledge when both are applicable[27]. To illustrate, suppose we believe the generalizations that mammals don't fly, bats fly, but injured bats do not. These can be represented as possible hypotheses. Add facts sufficient to capture the subclass information. If Dracula is an injured bat, we want to explain the prediction that he doesn't fly with the theory that injured bats don't fly rather than the theory that mammals don't fly or the prediction that he flies because bats typically do fly; we prefer the theory that uses the most specific information.
- Applied to the Yale shooting scenario[12,10], theory formation produces undesirable yet consistent theories that rankle intuition. (Other formalisms fare no better.) Goodwin[10] and Kautz[13] use *persistence* to choose the theory that is correct in the intended interpretation. (We note Loui's counterexamples.)

In some sense, these solutions try to provide a general way to *complete* predicates with additional assumptions.

The second class of comparators views consistent explanations as equals and compares them on a *heuristic* basis, in the sense the preferred explanation might not be the correct one. Since any explanation might be correct in the intended interpretation, we don't expect general ways to choose among them but rather require other kinds of knowledge about the explanations themselves. In our current research, we characterize this extra information as a valuation function which ranks competing explanations according to "goodness". Presently we are most interested in measuring the likelihood an explanation is true given some observations, but in other situations, we might compare explanations according to cost or virulence.

A current investigation of this second class of *heuristic* theory comparators has yielded promising initial results. Here we give Poole's semantics for theory formation and add semantics for heuristic theory comparators where observations are known to be true. We discuss properties a theory valuator must have so that a comparator is *useful.* In particular, probability is a useful theory comparator. We then describe a procedure that generates theories according to the order defined by the comparator. Lastly, we touch on a way to give semantics to theory valuators, (in particular, probability) with the aim of achieving efficiency but not at the cost of losing meaning.

2 Semantics

2.1 Theory formation

Here we give the formal semantics of the theory formation process.

The user provides the system with two sets of formulae:

F is a set of closed formulae called *facts.*

Δ is a set of formulae called *possible hypotheses.* These can be defaults, prototypes, possible malfunctions or anything else we are prepared to accept as part of an explanation of some goal.

A goal g is *explainable* is there is some D, a set of ground instances of members of Δ, such that

$$F \cup D \models g$$
$$F \cup D \text{ is consistent.}$$

and D is called an *explanation* of g.

If Δ is a set of generalised knowledge and g is some proposition we want to predict, we have default reasoning. If Δ contains possible diseases or malfunctions as well as defaults, and g is the observations of the system, we have diagnosis.

Poole describes his system in the framework of the first order predicate logic, but notes that the system is not restricted to that logic[19]. We want an explanation which predicts the goal, but does not predict anything which we know is false.

2.2 Theory preference

A *theory valuator* is a function $m(g, D)$ where D is an explanation of goal g. We assume the set of facts F fixed in any application. The range of m is a poset R ordered by $\preceq$. We *prefer* theory

D_1 to theory D_2 if $m(g, D_2) \preceq m(g, D_1)$ and we write $D_2 \preceq D_1$. If $D_1 \preceq D_2$ and $D_2 \preceq D_1$ then $D_1 \approx D_2$. If $D_1 \preceq D_2$ and not $D_1 \approx D_2$ then $D_1 \prec D_2$.

A theory valuator m is *useful* if:

1. m is defined for every D and g such that D is an explanation for g, i.e., whenever it needs to be.
2. if D_1 and D_2 are explanations for g, then $m(g, D_1) \preceq m(g, D_2)$ or $m(g, D_2) \preceq m(g, D_1)$.
3. if $F \cup D_2 \Rightarrow D_1$, then $m(g, D_2) \preceq m(g, D_1)$. This corresponds to preferring theories which make fewer assumptions, in the sense of logical implication or the subset relation (the above implication holds, in particular when $D_1 \subseteq D_2$). Following William of Occam, we always prefer a simpler explanation. Adding hypotheses decreases certainty or increases cost, both undesirable features.

These assumptions are consistent with Aleliunas' extension of the result of Cox[4,1]. Then it is easy to show that standard real-valued probability is a useful valuation function and m can be interpreted as the conditional probability of D given g. We retain the more general notion of theory valuation because it seems consistent with every heuristic theory comparator we can think of.

Below, we assume that m is a useful valuator. If D_k is such that there is no j such that $m(D_k, g) \prec m(D_j, g)$, then D_k is a *preferred theory* with respect to m and g and F. We usually simply call D_k the preferred theory when the context is clear.

3 Implementation

Poole et al. describe an implementation of explainability in [23] and include a terse Prolog implementation. The theorem prover tries to prove g from F and Δ, and makes D the set of instances of members of Δ used in the proof, as long as it fails to prove that $F \cup D$ is inconsistent. When the theorem prover uses a ground instance δ of Δ, we say it has assumed the hypothesis δ.

We use the same technique, but build all proofs incrementally using a separate process for each proof.

Each partial proof of the observations, carries a state parameter $< O, H, N >$ describing globally known observations, local partial explanations and the value $N$of the theory valuator for O and H. If we are ranking explanations by likelihood, N is an optimistic upper bound on the value of $m(O, H)$ when the proof halts. The procedure begins with the initial observations, the null hypothesis and an appropriate initial value for the null hypothesis. In the case of real-valued probabilities, this is 1. N decreases in any process that assumes a new hypothesis; it changes globally whenever new observations are made. (This happens during the consistency check as predictions of the theory are verified by making new observations by testing or asking the user.)

The procedure follows.

1. Initialize the state to $< g, \{\}, n >$ where g represents the initial observations, the empty set represents the null theory, and n is an appropriate initial value.
2. If a new hypothesis h is assumed, fork the process. If $F \cup D \not\models \neg h$, set the state of the parent process from $< O, D, N >$ to $< O, \{h\} \cup D, m(O, \{h\} \cup D) >$ and suspend; else kill the process. Make the child backtrack to the last choice point (i.e., "un-assume" h).

3. If a new observation o is made (if the user provides more information about the domain, reset the states of all processes from $< O, D, N >$ to $< \{o\} \cup O, D, m(\{o\} \cup O, D) >$. Suspend all processes.

4. If all processes are suspended, kill all processes with inconsistent $O \cup H$ and restart a process with a highest value of N. If no running or suspended processes remain, there is no solution, halt. Continue running the process until either a new fact is learned or a new hypothesis is assumed, then go to step 2. If a process completes a proof and any process may terminate, print the answer and halt.

This is completely undecidable, since the procedure defines a control structure for a theorem prover that does consistency checks. However, if it halts, it has either found the preferred explanation or there is no explanation.

As long as no new observations are made, step 2 ensures that we are always working on the most promising explanation. Step 3 ensures all proofs exploit any new knowledge.

We have a prototypical Prolog program which compiles specifications of F and Δ and m to a Prolog program that searches for a preferred theory. With no possible hypotheses and a null valuator, the compiled code behaves exactly like Prolog.

4 Theory valuators

We have given semantics for theory formation and theory preference given a theory valuation function, but we have not given semantics for the valuation function itself. We are investigating giving ways of semantics to theory valuators based on the theory formation formalism and give the basic ideas below with specific reference to probabilities.

The practical activity of calculating likelihood always seems to involve making statistical assumptions *in the absence of other information.* Otherwise the required calculations are intractable. For example:

- In the electronic circuit diagnosis problem, gates fail independently [5].
- Conditional independence of symptoms is commonly assumed in the medical diagnostic situation [2,17,14].
- The maximum entropy assumption. [3]
- When performing diagnosis, given no information to the contrary, we assume an individual might have contracted some disease with likelihood equal to the statistical mean unless we have information that the individual is particularly susceptible or particularly resilient to that disease.

In [11], Benjamin Grosof makes similar observations and places calculation of probabilities in the setting of Reiter's defaults and pointwise circumscription. He also discusses another independence assumption, maximization of conditional independence.

But it may not be natural to make any single statistical assumption an axiom of a theory of probability. A frequent problem with the old medical diagnostic systems, for example, is that a host

of statistical assumptions were forced on the data *a priori*, resulting in the loss of useful exceptional information [8,16]. No statistical assumption is foolproof in general and there may be situations where we want to make other kinds of assumptions.

Theory formation potentially gives us a simple solution to all these problems. We divide a theory of probability into facts F_p and *domain dependent* statistical assumptions Δ_p. This lets the user choose what statistical assumptions to make in a given setting. The assumptions are used only when no other information is known or can be derived. This also gives an interesting symmetry: probability as a useful theory valuator, can be used to guide the theory formation machinery to produce the preferred theory first and the theory formation machinery can be used to obtain the most meaningful probability in given situation.

Many uncertainty formalisms trade tractability for meaning. We believe that it is will be the special case where the expensive computation occurs in our system, and in general we can have the best of both worlds. In any case, this idea seems to accommodate any valuation functions meeting our usefulness criteria and allows arbitrary exceptional information. Practical ways to incorporate such statistical assumptions are currently under investigation.

5 Conclusions

We have borrowed from the ideas of Reiter[26], deKleer[5], Reggia[24], Genesereth[9,7], and Grosof[11] to construct a framework for theory formation independent of the logic used (though we favour first order predicate logic) and independent of the heuristic used to guide the theorem proving procedure to the preferred theory (though we favour probability and its derivatives).

We suggested that the nonmonotonicity of most heuristics used to guide theorem proving can be easily modelled with theory formation, providing an interesting duality to the theory.

Our implementation handles diagnosis of simple circuits (the full adder example from Genesereth), and small medical diagnostic problems. The initial results seem to suggest that probability and logic work together well as two simple different interacting cooperative processes; however, many interesting theoretical and practical problems remain to be solved.

6 Acknowledgements

We would like to thank members of the LPAIG for discussions and suggestions, especially Denis Gagne and Scott Goodwin for very constructive criticisms.

This research was supported by the Natural Science and Engineering Research Council of Canada and the Institute for Computer Research at the University of Waterloo.

References

[1] Romas Aleliunas. Mathematical models of reasoning based on abstract probability algebras. 1986. submitted.

[2] M. Ben-Bassat, R.W. Carlson, V.K. Puri, M.D. Davenport, J.A. Schriver, M. Latif, R. Smith, L.D. Portigal, E.H. Lipnick, and M.H. Wel. Pattern-based interactive diagnosis of multiple dis-

orders: the medas system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:148–160, 1980.

[3] Peter Cheeseman. A method of computing generalized bayesian probability values for expert systems. In *Proceedings 7th IJCAI*, pages 183–202, 1983.

[4] Richard T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14(1):1–13, 1946.

[5] Johan de Kleer. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[6] David W. Etherington and Raymond Reiter. On inheritance hierarchies with exceptions. In *Proceedings AAAI-83*, pages 104–108, 1983.

[7] J.J. Finger and Michael R. Genesereth. *RESIDUE: A Deductive Approach to Design Synthesis.* Technical Report HPP-85-1, Stanford Heuristic Programming Project, January 1985.

[8] Dennis G. Fryback. Bayes' theorem and conditional nonindependence of data in medical diagnosis. *Computers and Biomedical Research*, 11:423–434, 1978.

[9] Michael R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[10] Scott D. Goodwin and Randy G. Goebel. Applying theory formation to the planning problem. In F.M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, Morgan Kaufmann, Los Altos, Cal., September 1987.

[11] Benjamin N. Grosof. Non-monotonicity in probabilistic reasoning. In *Proceedings Uncertainty in Artificial Intelligence Workshop*, pages 91–98, August 1986.

[12] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logic and the frame problem. In *Proceedings AAAI-86*, pages 328–333, 1986.

[13] H. Kautz. The logic of persistence. In *Proceedings AAAI-86*, pages 401–405, 1986.

[14] P.M. Lewis. Approximating probability distributions to reduce storage requirements. *Information and Control*, 2:214–224, 1959.

[15] R.E. Mercer and R. Reiter. *The Representaiton of Presuppositions Using Defaults.* Technical Report 82-1, University of British Columbia, Dept. of Computer Science, 1982.

[16] Marija A. Norusis and John A. Jacquez. Diagnosis. i. symptom nonindependence in mathematic models for diagnosis. *Computers and Biomedical research*, 8:156–172, 1975.

[17] Yun Peng and James A. Reggia. A probabilistic causal model for diagnostic problem-solving. part one: integrating symbolic causal inference with numeric probabilistic inference. 1986. submitted.

[18] C. Raymond Perrault. An application of default logic to speech act theory. unpublished draft.

[19] David Poole. Building consistent theories. 1986. submitted.

[20] David L. Poole. *Default Reasoning and Diagnosis as Theory Formation.* Technical Report CS-86-08, University of Waterloo Department of Computer Science, 1986.

[21] David L. Poole. On the comparison of theories: preferring the most specific explanation. In *Proceedings 9th IJCAI*, pages 144–147, 1985.

[22] David L. Poole. The use of logic. 1986. submitted.

[23] David L. Poole, R.G. Goebbel, and R. Alelunias. Theorist: a logical reasoning system for defaults and diagnosis. In Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, Springer-Verlag, New York, 1987.

[24] James A. Reggia, Dana S. Nau, and Pearl Y. Wang. Diagnostic expert systems based on a set covering model. *Journal of Man-Machine Studies*, 19:437–460, 1983.

[25] Raymond Reiter. A logic for default reasoning. *Artifical Intelligence*, 13:81–132, 1980.

[26] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[27] D. S. Touretzky. Implicit orderings of defaults in inheritance systems. In *Proceedings AAAI-84*, pages 322–325, 1984.

[28] Paul van Arragon. User models in theorist. 1986. Ph.D. Thesis Proposal, Department of Computer Science, University of Waterloo.

The Multiple Extension Problem Revisited

Denis Gagné

Logic Programming and Artificial Intelligence Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario. CANADA
N2l 3G1
jddgagne@waterloo.CSNET

April 24, 1987

Abstract

The aim of this paper is threefold. First we present evidence to the effect that: in a multiple extension context, when using our common sense to reason about problems that use logically equivalent axiomatisation, we sometime come to different conclusions.

Second, we present an extension to the Theorist framework, which provides us with a new way to look at defaults by allowing us to explicitly state exceptions to them when making predictions. Finally we argue, based on the previous result, that the multiple extension problem either arises from an incomplete axiomatisation (misrepresentation) of the situation or is not a problem at all.

1 Introduction

Without loss of generality *vis a vis* other formalisms of non-monotonic reasoning, namely, the three best known or influential ones: *Default Logic* [Reiter 80], *Circumscription* [McCarthy 86] and *Non-Monotonic Logic* (NML) [McDermott & Doyle 80], we will present, and base our arguments in this paper on examples using the *Theorist* framework [Poole et al. 86].

The Theorist framework views reasoning as constructing theories that logically imply a set of observations. The sets of formulae that are distinguished within the framework are:

F: The set of facts. This is a consistent set of formulae(*wff*), true of the domain being studied.

Δ: The set of possible hypotheses. These formulae(*wff*) specify sentences that we are ready to assume to be generally true.

G: The set of observations.

Definition: An observation g is **explainable** using theory D, a set of ground instances of elements of Δ, if

$$F \cup D \models g, \text{ and}$$
$$F \cup D \text{ is consistent.}$$

2 Multiple Extensions

In order to properly discuss the multiple extension problem, we must first clarify the concept of an *extension.* Informally, an *extension* is a set of sentences that "extend" the sentences in F according to the default situation being studied.

Definition: An **extension** E of a set of sentences F, is a minimal set of sentences that satisfy the following properties:

1) E contains F
2) E is closed under theoremhood
3) $d \in E$, if d is a ground instance of an element $\in \Delta$, and $\neg d \notin E$.

A default situation has one or more extensions, each of which are to represent a coherent and consistent state of the domain.

We would like to concentrate here on a special kind of extension: mutually contradictory extensions. Of particular interest are contradictory consequences that arise from defaults with distinct, but not mutually exclusive, bodies. In many of these cases, one of the extensions of the default situation gives rise to an answer, and the negation of this answer is obtained by another extension of the same situation.

3 Mutually Contradictory Extensions

Now let's set the stage for later discussion by presenting what appear to be straightforward axiomatisations of three simple situations using statements that we generally assume to be true.

Situation 1

$$\Delta_1 = \{animal(X) \wedge tamed(X) \supset doTricks(X), \quad tiger(X) \supset \neg doTricks(X)\}$$

$$F_1 = \{tiger(X) \supset animal(X)\}$$

With the intended interpretation of this axiomatisation being that:
1) generally animals that are tame do tricks
2) generally tigers don't do tricks
and we know as a fact that tigers are animals.

Situation 2

$$\Delta_2 = \{bird(X) \wedge alive(X) \supset fly(X), \quad penguin(X) \supset \neg fly(X)\}$$

$$F_2 = \{penguin(X) \supset bird(X)\}$$

With the intended interpretation that:
1) generally birds that are alive fly
2) generally penguins don't fly
and stating the fact that penguins are birds.

Situation 3

$$\Delta_3 = \{painting(X) \wedge rembrandt(X) \supset like(X), \quad paintingOfLandscape(X) \supset \neg like(X)\}$$

$$F_3 = \{paintingOfLandscape(X) \supset painting(X)\}$$

With the intended interpretation that:
1) generally I like paintings by Rembrandt
2) generally I don't like paintings of landscapes
and acknowledging the fact that paintings of landscapes are paintings.

The axiomatisation given in each case appears to be adequate and to capture the intended interpretation for each of the situations. Note the fact that these three axioma-

tisations have the same syntactic structure, and as a result, one would expect them to be logically equivalent as well (i.e., the corresponding extension is the intended one for each of them).

Now consider in situation 1 the observation of *panta*, a tamed tiger. Using common sense, one would agree with the conclusion that *panta*, being tamed, is able to do tricks. On the other hand, in situation 2, when observing the fact that *tweety* is a penguin and that *tweety* is alive, one would come to the conclusion that *tweety* will not fly since *tweety* is a penguin. For clarity we will call these conclusions, from our common sense, the "*intended answer*," for each of the situations.

Expressed in formulae, the observation of *panta* being a tamed tiger in the first situation corresponds to the addition of the facts:

$$\{tamed(panta), tiger(panta)\}$$

to F_1. In the second situation the facts added to F_2, when observing that *tweety* is alive, and is a penguin are:

$$\{alive(tweety), penguin(tweety)\}$$

Notice two important facts. First that even though situation 1 and 2 use axiomatisations that have the same syntactic structure, our common sense leads us to opposite conclusions, namely that *panta* is able to do tricks but that *tweety* does not fly.

Second, that from the given axiomatisations, mutually contradictory extensions arise for both situations. As a result, for Theorist, it is "unknown" given the axiomatisations which answer is the proper one, since we can explain (conclude) both the intended answer and its negation. Derivations of the intended answers and their negations are shown in fig. 1 for situations 1 and 2. Our framework seems to fail to completly capture our common sense in these cases.

We claim that when coming to the intended answers using our common sense for situations 1 and 2, we actually used some extra knowledge that is not explicitly stated in the given axiomatisations. Even though we agreed earlier that the axiomatisation given seemed to have adequately captured the situations, we believe that some knowledge was missing, which as a result led Theorist to be unable to predict only the intended answer and not its negation. It is then the axiomatisation that fails to capture our common sense and not our framework.

To further illustrate our point, and to try to isolate the extra knowledge used, let us now turn to situation 3, and add the following facts to F_3:

$$\{paintingOfLandscape(theGorge), rembrandt(theGorge)\}$$

conserving the same syntactic structure of the axiomatisations for the three situations. In situation 3, when observing this painting titled *The Gorge* featuring a landscape and

---

Situation 1

* $doTricks(panta) \leftarrow animal(panta) \wedge tamed(panta)$
$animal(panta) \leftarrow tiger(panta)$
$tiger(panta)$
$tamed(panta)$.

$\neg doTricks(panta) \leftarrow tiger(panta)$
$tiger(panta)$.

Situation 2

$fly(tweety) \leftarrow bird(tweety) \wedge alive(tweety)$
$bird(tweety) \leftarrow penguin(tweety)$
$penguin(tweety)$
$alive(tweety)$.

* $\neg fly(tweety) \leftarrow penguin(tweety)$
$penguin(tweety)$.

Figure 1: Consistent derivations of the intended answer(*) and its negation for situations 1 and 2

signed by Rembrandt himself, most people will give the answer "I don't know" when asked to conclude wheter I like this painting or not, and will add "Well, it depends on how much more you like Rembrandt's paintings than you dislike landscape paintings."

This "unknown" intended answer reinforces our claim that extra knowledge, not explicitly stated in the axiomatisations, was required to obtain the intended answers for situations 1 and 2. Furthermore, the extra comment given, indicates that this extra knowledge is concerned with the way in which we use these statements, that we assume to be generally true, to make predictions.

Our intuition is that this extra knowledge is actually of the general form: "*in this particular case I am no longer ready to assume this default to be true when making predictions.*"

4 The Required Extra Knowledge

What we need, are logically founded criteria to restrict the use of our defaults when it is desired. According to our intuition, in situation 1, we would like to add the knowledge that: "*in the particular case of being tamed I am no longer ready to assume that tigers will not do tricks.*" In situation 2, we would add the knowledge that: "*in the particular case of penguins I am no longer ready to assume that birds fly.*" Depending on your own taste you would, in situation 3, either add the knowledge that:"*in the particular case of paintings of landscapes, I am no longer ready to assume that I like Rembrandt's paintings, when make predictions*" or that: "*in the particular case of Rembrandt's paintings, I am no longer ready to assume that I don't like paintings of landscapes, when making predictions.*"

One way to add this extra knowledge to the axiomatisation, and realising our goal (obtaining the intended answer) is to name our defaults, and have facts that restrict their use when making prdictions for particular cases. Naming the defaults in the Theorist framework has the following syntax and semantics[1]:

syntax:

$$n_i(\mathbf{X}) : f_i[\mathbf{X}]$$

Where n_i is a *predicate symbol*, the unique name for default $f_i \in \Delta$, and f_i is a *wff*. n_i is parameterised by $\mathbf{X} = X_1 \ldots X_n$, the free variables in f_i. When there are no free variables, the default is then said to be **closed** [Poole 86a].

semantics: Given F and Δ as defined earlier

$$\Delta' \equiv \{n_i(\mathbf{X}) | f_i[\mathbf{X}] \in \Delta\}$$
$$F' \equiv F \cup \{n_i(\mathbf{X}) \supset f_i[\mathbf{X}] | f_i[\mathbf{X}] \in \Delta\}$$

[1]Thanks to David Poole for indicating the proper formal definition of naming defaults.

In this modified version of Theorist, Δ' contains the names of the defaults from Δ. The logical relation between the names, and the defaults they are associated with, are added to F to form F'.

We have established above how our defaults can be properly named. We now have to define logically founded criteria that restrict their use in particular cases when making predictions. To achieve this we introduce a new set of formulae to be distinguished in the Theorist framework along with F' and Δ'.

F' and Δ': The named version presented above.

∇: The restrictive facts. These formulae(*wff*) specify exceptions to the defaults in Δ'.

We then have the following new definition of explainable:

Definition: An observation g is **explainable** using theory D, a set of ground instances of elements of Δ', if

$$F' \cup D \models g, \text{ and}$$
$$F' \cup D \text{ is consistent.}$$

We also introduce a seperate definition for the concept of predictions.

Definition: A theory D (a set of ground instances of elements of Δ') **predicts** g if

$$F' \cup D \models g, \text{ and}$$
$$F' \cup D \cup \nabla \text{ is consistent.}$$

In this extended version of Theorist, we distinguish between explanations and predictions. Having observed a fact one may ask Theorist to explain this observation, on the other hand one may also ask Theorist if, based on its knowledge, it can predict a certain conclusion.

5 Implementation

An interpreter for Theorist implemented in Prolog is currently being used at the University of Waterloo based on a complete theorem prover. Its implementation is discussed in detail in [Poole et al. 86]. Augmenting Theorist with the ideas presented in the previous section is very simple, and only requires minor changes.

Theorist already has a compiler that compiles facts and defaults into Prolog statements. The new version accepts named defaults with the form; $n(\mathbf{X}) : f[\mathbf{X}]$, and compiles these

statements into; $n(\mathbf{X}) \in \Delta'$ and $n(\mathbf{X}) \supset f[\mathbf{X}] \in F'$ according to the given semantics. The compiler is also augmented to accept formulae that are elements of ∇.

The next change consist of modifying the interpreter to make predictions. The same algorithm as for explanation can be used except that, anytime consistency was verified for explanation, now the elements of ∇ are also considered for prediction. These modifications have been made, and the examples that follow were verified using the new version.

Shown in fig. 2 are the named versions of situations 1 and 2 along with the restrictive facts based on the intuitions given a the beginning of the previous section. Note that in this new Theorist framework, only the intended answer is predicted[2].

6 Discussion

The modifications presented for Theorist introduce in a clear and simple way knowledge that is required for any framework wanting to achieve common sense reasoning,that is, exceptions to defaults when making predictions.

Before we go further, we would like you to notice that given the axiomatisations for situations 1 and 2, it is possible to derive results that are unexpected. When using both defaults, one as presented and the contrapositive of the other, it is possible to conclude by contradiction in situation 1 that $\neg penguin$ is true for any individual that is not explicitly known as being a *penguin*, and in situation 2, that $\neg tiger$ is true for any individual that is not known to be one. These conclusions by contradiction are certainly undesirable as predictions. All that is required to fix this, is to add a restrictive fact to ∇ that state that both defaults can't be used for the same individual:

$$\{default1(X) \supset \neg default2(X)\}$$

It then becomes inconsistent to predict $\neg penguin$ and $\neg tiger$.

When completing an axiomatisation by adding restrictive facts to ∇, one must be careful not to include subclasses that are not exceptions, with their parent class. Therefore completing the axiomatisation seems to only be valid for the particular situation with the given axiomatisation, if the situation or the axiomatisation are modified, the restrictive facts added to complete the axiomatisation may show themselves to be incomplete or to lead to over-restricted results. In these cases a different set of restrictive facts are required.

Restricting extensions in *Non-Monotonic Logic* has been presented by Brewka [Brewka 86]. He adds a standard predicate *APPL* to each default, using the format:

$$\mathbf{M}\ APPL(rulename, variables)$$

[2] An inconsistency arises with elements of ∇ when trying to verify if Theorist predicts the negation of the intended answer.

Modified Situation 1

$$\Delta_1 = \{default1(X),\ default2(X)\}$$

$$\begin{array}{lll} F_1 = \{default1(X) & \supset & [animal(X) \wedge tamed(X) \supset doTricks(X)], \\ default2(X) & \supset & [tiger(X) \supset \neg doTricks(X)], \\ tiger(X) & \supset & animal(X), \\ tiger(panta), & & \\ tamed(panta) & & \} \end{array}$$

$$\nabla_1 = \{tamed(X) \wedge tiger(X) \supset \neg default2(X)\}$$

Modified Situation 2

$$\Delta_2 = \{default1(X),\ default2(X)\}$$

$$\begin{array}{lll} F_2 = \{default1(X) & \supset & [bird(X) \wedge alive(X) \supset fly(X)], \\ default2(X) & \supset & [penguin(X) \supset \neg fly(X)], \\ penguin(X) & \supset & bird(X), \\ penguin(tweety), & & \\ alive(tweety)\} & & \} \end{array}$$

$$\nabla_2 = \{penguin(X) \supset \neg default1(X)\}$$

Figure 2: Modified situations 1 and 2

in order to explicitly control the applicability of default rules in that framework. In circumscription, the control of the application of defaults can be found in what is called "*cancellation of inheritance axioms*" [McCarthy 86] but is done in a less intuitive way (side effect of circumscribing *ab z*). Etherington [Etherington 87] proposed to use seminormal defaults, in *Default Logic*, in order to achieve the same goal. He uses inference rules of the form:

$$\frac{A(X) : \mathbf{M}\ B(X) \wedge \neg C_1(X) \wedge \ldots \wedge \neg C_n(X)}{B(X)}$$

meaning that: it is required that the case under consideration not be known to be an exceptional case, $C_1 \ldots C_n$, for this inference to take place.

We acknowledge the fact that we could have obtained the intended answers using a different approach. For example in the old Theorist, using pertinent preference criteria, we could "prefer" an extension over others to obtain the intended answer. This criterion could be a probability theory or any other kind of criteria that prefers the intended answer for a given context.

We believe that completing the axiomatisation so that undesirable predictions can't be made, as we proposed, is a simpler, and more intuitive way to achieve our goal. We ourselves seem to use a similar form of reasoning:"*I believe that generally birds fly, but penguins are one of the exceptions to this rule.*" We often use a similar format to guide ourselves in different tasks, for example in grammar we have generally applicable rules with exceptions.

A generic preference criterion, along the lines of the one proposed by Poole [Poole 85] to obtain the intended answer, appears to be impossible to find given that the above three examples are equivalent but have different intended answers. According to Poole, no "Theory" is strictly more specific than the other, for each of the three situations presented.

When using a preference criterion to select the intended answer, the multiple extensions are still present, it is just that one is preferred over the others. A problem with the "preference method" is that since no generic criterion is possible, the preference criteria are based on particular contexts (e.g., chronological persistence for temporal situations [Goebel & Goodwin 86]). In a more complicated situation, one would need a knowledge base of statements of the form:

$$applicable(criterion, context)$$

to cover the vast number of possible contexts, and the system would have to recognize (which is very unlikely) the particular context under consideration in order to know which criterion to apply. If an axiom has to be added to the axiomatisation in order to identify the context of the default situation at hand, then the "preference method" becomes very similar to what is proposed here.

The Yale Shooting Problem

$$\Delta = \{frame(S, A, F)\}$$

$$\begin{aligned} F = \{&frame(S, A, F) \supset [hold(F, S) \leftrightarrow hold(F, do(A, S))], \\ &hold(loaded, S) \supset \neg hold(alive, do(shoot, S)), \\ &hold(loaded, do(load, S)), \\ &hold(alive, s_0)\} \end{aligned}$$

$$\nabla = \{\neg frame(S, shoot, alive)\}$$

Figure 3: The Yale Shooting Problem Solved

The attractiveness of the framework presented here is its simplicity, and the ease with which we can represent common sense knowledge within the framework.

7 Do we Have a Problem?

Hanks and McDermott [Hanks & McDermott 86] argued, based on an example "The Yale Shooting Problem," that non-monotonic reasoning was unable to capture our common sense, since contradictory answers were derivable for the considered problem. We will argue that Hanks and McDermott's axiomatisation of the Yale Shooting Problem, which led them to this conclusion, was incomplete, and show how to complete it.

Loui [Loui 86] presents an axiomatisation of a different situation that has the same syntactic structure as the Yale Shooting Problem, but has a contradictory intended answer. You can see, based on Loui's example, that the axiomatisation of the Yale Shooting Problem given by Hanks and McDermott, seems to fall into the same pitfall as the *tamed tiger*, and *penguin* examples.

Their axiomatisation suffers the same problem as the examples given before; some extra knowledge restricting the use of defaults for particular cases is required to properly make predictions. Shown in fig. 3 is an axiomatisation of the Yale Shooting Problem, using the framework presented in this paper, where contradictory answers can't be predicted.

The extra knowledge we added in ∇ to solve the Yale Shooting Problem states that "*when there is shooting going on, I am no longer ready to assume,in my predictions, that the person remains alive.*" Given the described situation in [Hanks & McDermott 86], this knowledge is intuitive and required. Using this axiomatisation we can predict that the person will not be alive after loading the gun, waiting and shooting, but in the event of the

person being observed alive after this sequence of actions we can build a theory to explain this situation.

There exist situations (e.g., the Quaker-Republican example (below) or situation 3) where it seems to be hard to find the proper restrictions on the use of defaults in predictions. In these situations the multiple extensions are not a problem because they represent the indecision that arises using our common sense to decide what the intended (proper) answer is. Once we set our mind on what the intended answer really is, it is then easy to add the proper restrictions in order to obtain the desired answer.

The Quaker-Republican Example

$$\Delta = \{default1(X), default2(X), default3(X)\}$$

$$\begin{aligned} F = \{&default1(X) \supset [quaker(X) \supset pacifist(X)],\\ &default2(X) \supset [pro-defence(X) \supset \neg pacifist(X)],\\ &default3(X) \supset [republican(X) \supset pro-defence(X)],\\ &quaker(nixon),\\ &republican(nixon)\} \end{aligned}$$

$$\nabla = \{\}$$

The intended interpretation of the axiomatisation of the Quaker-Republican example is:
1) generally quakers are pacifists
2) generally people that are pro-defence are not pacifists
3) generally republicans are pro-defence
And we observe Nixon, a quaker and also a republican. It is undecided whether Nixon is a pacifist or not.

8 Summary and Conclusions

We started by showing, for a specific kind of default situation having axiomatisations with the same syntactic structure, that using our common sense, we sometimes pick a different extension to be the intended one. From there, we suggested that mutually contradictory extensions arise from an incomplete axiomatisation, and showed how one could go about recovering from it, using a simple framework with clear semantics. The solution presented is to name defaults, and add facts that restrict their use in particular cases of predictions. The solution was shown to be adequate for the given examples.

Based on these results, we finally argued that the multiple extension problem either arises from an incomplete axiomatisation or is not a problem at all. We believe that using the examples introduced in this paper, and Yale Shooting Problem, we have shown that the multiple extension problem only arises because we form intended answers base on extra knowledge that is not made explicit in the axiomatisation.

We have shown that when this extra knowledge is made explicit in the axiomatisation the perceived problem disappears. We finally pointed out that, when there is indecision from our common sense about an answer, the multiple extensions represent this indecision, and in these cases, are not a problem at all.

Acknowledgements

We would like to thank Bruce Kirby, and Eric Neufeld for proposing the examples that sparked the ideas in this paper, and David Poole for his assistance in formalizing the framework. Very special thanks goes to Scott Goodwin for his support and help during this work. This research was done under a Natural Science and Engineering Research Council award.

References

[Brewka 86] Brewka, G., Tweety Still Flying:Some Remarks on Abnormal Birds, Applicable Rules and a Default Prover. In *Proceedings of AAAI-86*, 1986, 8-12.

[Etherington 87] Etherington, D.W., Formalizing Nonmonotonic Reasoning Systems. In *Artificial Intelligence Journal*,31, 1987, 41-85.

[Goebel & Goodwin 86] Goebel, R.G. and Goodwin, S.D., Applying Theory Formation to the Planning Problem. In *Proceedings of the 1987 Workshop on the Frame Problem in AI*, Morgan Kaufmann, 1987, 207-.

[Hanks & McDermott 86] Hanks, S. and McDermott, D., Default Reasoning, Nonmonotonic Logics, and the Frame Problem. In *Proceedings of AAAI-86*, 1986, 328-333.

[Loui 86] Loui, R.P., Response to Hanks and McDermott: Temporal Evolution of Beliefs and Beliefs about Temporal Evolution. To appear in *Cognitive Science*.

[McCarthy 86] McCarthy, J., Applications of Circumscription to Formalizing Common-Sense Knowledge. In *Artificial Intelligence Journal*, 28, 1986, 89-116.

[McDermott & Doyle 80] McDermott, D. and Doyle, J., Non-Monotonic Logic I. In *Artificial Intelligence Journal*, 13, 1980.

[Poole 85] Poole, D.L., On the Comparison of Theories: Prefering the Most Specific Explanation. In *Proceedings of IJCAI-85*, 1985, 144-147.

[Poole 86a] Poole, D.L., Default Reasoning and Diagnosis as Theory Formation. Department of Computer Science, University of Waterloo, Tech. Rep. CS-86-08.

[Poole et al. 86] Poole, D.L., Goebel, R.G. and Aleliunas R., Theorist: A Logical Reasoning System for Defaults and Diagnosis. Department of Computer Science, University of Waterloo, Tech. Rep. CS-86-06, 1986.

[Reiter 80] Reiter, R., A Logic for Default Reasoning. In *Artificial Intelligence Journal*, 13, 1980.

Prolog with Loop Checking

Paul A. Strooper

CS 786 Project
Winter 1987

ABSTRACT

Several attempts have been made to prevent Prolog from looping. Most of these are based on the idea of blocking certain derivations in which the current goal matches a goal higher in the proof tree. Great care has to be taken in doing this so that we do not eliminate any of the answers.

In this paper we suggest an approach which is based on using the standard Prolog inference rule for all the goals which are not instances of goals higher up the proof tree. For those goals, we apply forward chaining using the answers generated for the higher goals. We prove this method is correct (sound) and also complete.

1. Introduction

Even though the search space as defined by SLD resolution[1] is complete (see [Lloyd84]), it is easy to generate infinite loops in Prolog. The reason for this is that the Prolog computation rule uses a depth first strategy, together with selecting the leftmost atom in a goal. Moreover, Prolog can go into an infinite loop, even when there exist more finite proofs (i.e. before all answers are found). An example of this is the following program:

(1) t(a,b);
(2) t(b,c);
(3) t(c,d);
(4) t(X,Z) ← t(X,Y) t(Y,Z);

If we now ask the system for all pairs (X,Y) for which t(X,Y) is true, the system will generate the following answers:

t(a,b);
t(b,c);
t(c,d);
t(a,c);
t(a,d);

after which it goes into an infinite loop. It never generates the answer t(b,d). Rearranging the rules would make things worse, as no answer would be generated at all if we put rule (4) as the first one.

Loop checking is an attempt to detect when we are going down an infinite branch of the search tree. Although this problem is equivalent to the halting problem and as such unsolvable in general ([Lewis81]), we can detect if we are on a possibly infinite branch for a large number of cases. A necessary (but not sufficient) condition for this to happen is that there is some predicate which occurs infinitely often on the particular branch. It thus seems desirable to detect when the current goal on a branch "matches" a goal higher up the same branch. The most useful case is when the current goal is an instance of the goal higher up. We will take this as a starting point for detecting loops.

---

[1] SLD resolution stands for *linear* resolution for *definite* clauses with *selection* function.

Earlier attempts to solve this problem suggested simply eliminating all branches in which the current goal is an instance of a higher goal (see [Covington85a]). Later it was shown that this simple method does not work ([Covington85b]), and additional conditions were suggested which have to be satisfied in order to be able to eliminate a branch ([Covington85b] and [Nute85]). Again, it was shown that there were cases for which this does not work ([Poole85]). The problem with this method is that it is very hard to prove that the elimination of a branch does not eliminate an answer. And in fact Brough and Walker show that any terminating top-down, left-to-right interpreter misses some answers ([Brough84]). Therefore they suggest that we should be looking at interpreters with a bottom-up component. In [Poole85] it is suggested that we should be modifying our programs to avoid the problem, rather than trying to find a rule which would accomplish the same thing, but which is applicable in very few cases.

Instead of modifying the program we suggest yet another way of looking at the problem. Our method is based on using program transformation techniques and forward chaining (the bottom-up part) on those goals which are instances of goals higher up the branch to solve the problem. We thus can use the work done on program transformation by Tamaki and Sato ([Tamaki83]) and the correctness of unit resolution ([Chang73]) to prove the correctness and completeness of our method[2].

After giving some of the definitions used in this paper, we will justify why it is useful to look at loop checking in Prolog. Then we will give an algorithm which behaves correctly if certain conditions are met, in the sense that all the answers it gives are correct and none of the answers are eliminated. After proving this correct behavior, we will suggest a more general method which has the same properties but can be used in more cases.

2. Definitions

Before we start discussing the system, we will first define some of the terms which are used throughout this paper.

We say a goal G_1 is an **instance** of a goal G_2, is there exists a substitution θ for the variables in G_2, such that $G_1{=}G_2\theta$. Although this is the usual definition, there are two things to note. First of all, throughout this paper we use the definition of instance without the *occur check* ([Lloyd84]). For example, we consider t(X,X) to be an instance of t(X,Y). Secondly, when we talk about a goal being an instance of a goal higher up the tree, we do not take any substitutions into account. For example, if the higher goal is $p(X)$ and we resolve this goal using the rule $p(a)\leftarrow p(Y)$, then we consider $p(Y)$ to be an instance of the higher goal $p(X)$, even though after the resolution the substitution $X = a$ has taken place (and thus the higher goal we are solving "has become" p(a)).

Starting with a program P, if there exists a successful derivation of the query[3] $?G$ with θ being the composition of all substitutions in the derivation, we say that $\forall(G\theta)$ is an **unconditional answer** (where the universal quantification is over all variables appearing in $G\theta$). In this case we know the answer is a logical consequence of the program P, which we write as:

$$P \models \forall(G\theta)$$

Suppose, that starting from the query $?G$, we have derived a new set of goals $G_1 \cdots G_n$ $(n \geq 1)$, with θ being the composition of all substitutions. Then the clause:

$$\forall\ (G\theta \leftarrow G_1 \cdots G_n)$$

is a **conditional answer** of the program, and again we have:

$$P \models \forall\ (G\theta \leftarrow G_1 \cdots G_n)$$

---

[2] Correctness here means that the answers which are produced are logical consequences of the program, and completeness refers to the fact that we do not eliminate any answers. In previous work the correctness part was always satisfied, however completeness was not guaranteed.

[3] The system only works for queries consisting of a single goal. If a query consists of more than one goal $G_1 \ldots G_n$, we can add the rule $G \leftarrow G_1 \ldots G_n$ to the program, and use the system on the query with the single goal G.

For a program P and a query Q, we define a **frontier** to be a complete set of unconditional and and conditional answers for a derivation of Q in P, such that all or-choices are in the derivation tree. In [Cheng87] a general correctness result is proven for a set of frontiers. The correctness we will use is a variation of this, and we will prove it using the correctness result for the unfolding rule as proven in [Tamaki83].

Unfolding is a technique which is widely used in program transformation ([Burstall75], [Tamaki83] and [Cheng87]). If C is a clause in program P with an arbitrary goal G in its body, and $C_1,...,C_n$ are all the clauses in P whose heads unify with G. Then the result of **unfolding** goal G in clause C are the clauses $C'_1,...,C'_n$, where C'_i is obtained from C by resolving goal G in the body with rule C_i(i.e using a one step resolution).

It is important to note that all rules have to be used when unfolding, and similarly all the answers have to be collected when calculating the frontier. For example, consider the program:

p(X) ← q(X);
p(a);
p(b);
q(X) ← p(X);
q(c);

and the query ?p(X). Unfolding this goal once we obtain the frontier with the following answers:

p(X) ← q(X);
p(a);
p(b);

Unfolding the goal q(X) in the body of the conditional answer gives us the new frontier:

p(X) ← p(X);
p(a);
p(b);
p(c);

Forward chaining is a special case of resolution. When we forward chain on a goal G in the body of a rule R using a fact F, we simply resolve G using F. For example, when we forward chain on the first goal in the body of the rule:

t(X,Y) ← t(X,Z) t(Z,Y);

using the fact t(a,b), we obtain the new rule:

t(a,Y) ← t(b,Y);

3. The Need for Loop Checking in Prolog

Prolog is one of the most widely accepted and used theorem proving systems. For example, it is used to implement the Theorist reasoning system ([Poole86]). Such reasoning systems would benefit from a loop detection algorithm. However, as was pointed out, currently there are no satisfactory loop detection algorithms for Prolog itself. Therefore we should first find such an algorithm for Prolog. After this is done, we can try to incorporate it into other reasoning systems and/or theorem proving systems.

One of the major differences with more general loop detection algorithms for graph searching algorithms is that it is hard to define when we are looping. The main reasons for this are the use of variables and the unification that takes place. In fact, we know that the problem of deciding if we are on an infinite branch is undecidable, since it is equivalent to the halting problem. Therefore, as is pointed out in [Kowalski79], there is no limit to the extent to which we can improve the ability of a theorem prover (such as Prolog) to detect loops. So the best we can do is to attempt to detect this in as many cases as possible. In doing this we should also keep the efficiency of the solution in mind. For example we can guarantee to find all answers to a query in Prolog, by examining the search space breadth first. However, due to the exponentially growing size of the search space this is not a very practical solution.

Keeping all this in mind, the best solution seems to be to use the efficient depth first approach (the left first part of the strategy of Prolog is not as important as far as efficiency goes) as much as possible, and still attempt to detect when this approach could lead us to an infinite branch. As was mentioned before, a first indication of this is that the current goal we are trying to resolve is similar to one higher up the same branch.

A first approach thus could be to stop going down a branch if the current goal has the same predicate symbol as one higher up that branch. However, as was shown in [Poole85], we cannot simply eliminate this branch from the search tree. Brough and Walker ([Brough84]) show that we need some other way of proving these goals if we want to guarantee that all answers are produced. It thus seems reasonable to use the work done for the higher goals to prove the current goal. If the goal is more general or does not unify with the goal higher up the branch, this cannot be done efficiently. On the other hand, when the goal is an instance of the higher goal, we know that any potential answer for the current goal certainly must also be an answer of the higher goal.

This leads us to the idea of stopping a derivation when the current goal is an instance of a goal higher up the tree, and use the previous answers produced for the higher goal to attempt to prove the lower one using forward chaining. This agrees with the idea mentioned in [Kowalski79], which says that at every stage we should choose the direction (forward or backward) of inference which gives rise to the least number of alternatives. For the first stage we can use the standard Prolog interpreter with a simple loop check. We also need to keep track of all the answers produced for the higher goal. For the second stage we can use these answers to resolve the lower goals. The next section gives a more precise description of the algorithm which implements this strategy.

Some programs cannot be handled by this strategy. This occurs if we have programs that build up structures, such as in:

leq(X X);
leq(X Y) ← leq(s(X) Y);

Starting with any ground query which is not a logical consequence of the program (such as ?leq(s(0) 0)), we would still go into an infinite loop. This is because no goal is an instance of a goal higher up the branch. Despite these types of programs for which our method does not behave correctly, there are many other cases for which it is useful. As was pointed out in [Covington85b], it is desirable to have loop checking in programs with transitive and/or symmetrical relations, and in programs which use biconditionals. Our method handles all these types of programs. Moreover, our method works correctly, even if the recursion is indirect, which was not the case for some of the earlier proposed solutions to the problem of loop checking (see [Nute85]).

4. Initial Version of Loop Checking

In this section we will suggest an initial loop checking algorithm which works correctly if the only loops that can occur are caused by the predicate which appears in the query. At the end of this section we show how the system works on two examples.

4.1. Conditions for the Algorithm

For a program P and a query Q, the algorithm suggested in the next section works correctly and does not eliminate any answers, if the following conditions hold:

(1) Every infinite branch of any subtree of the search tree rooted at Q is such that there is a goal on that branch which is an instance of a goal higher up that branch.

(2) Every infinite branch of any subtree of the search tree rooted at Q is such that it has a goal appearing on it which is an instance of Q.

(3) Any goal appearing in the body of any clause of P with the same predicate symbol as Q, is in fact an instance of Q.

In the next section we will show why the third condition is actually not needed. However, we include it here, because the proof of completeness becomes a lot simpler this way. In section 6 we will show that we can also eliminate the second condition by using a more general algorithm.

4.2. Algorithm 1

The algorithm is based on using both SLD-resolution with the Prolog selection function, and forward chaining on those goals which are instances of goals higher up the branch of the proof tree, to find answers to a query. During one stage of the algorithm, we use SLD-resolution to find a set of conditional and unconditional answers for the query. Moreover, all the unconditional answers are such that the first goal in the body is an instance of the original query. During the second stage of the algorithm, we resolve the first goal of the bodies of all the conditional answers using all the unconditional answers.

To make the algorithm as efficient as possible we have to perform the forward chaining in such a way that we do not resolve a goal with the same fact more than once. We also have to resolve every goal which is an instance of the original goal with every answer found so far. To accomplish this, we keep lists of conditional and unconditional answers which already have been resolved (*old_ca* and *old_ua*), and we put any newly found answers in two separate lists (*new_ca* and *new_ua*). Now we only need to resolve the goals for the new conditional answers using both the new and old unconditional answers, and those for the old conditional answers using the new unconditional answers. The list *to_unfold* is used to keep answers which have yet to be unfolded.

Given a query Q and a program P, the algorithm is:

Step 0: Set $old\_ca = new\_ua = new\_ca = old\_ua = \{\}$, and let *to_unfold* be those clauses of P whose heads unify with Q.

Step 1 (Unfolding): For each clause in *to_unfold*, use SLD-resolution with the standard Prolog computation rule (leftmost and depth-first), to unfold the goals in the body. Do this while there are goals left, and while the first goal in the body is not an instance of the original query Q. Put all (the system backtracks whenever an answer is found to make sure all answers are collected) the conditional and unconditional answers that were found in all of these frontiers (there is one frontier for each of the clauses in *to_unfold*) in *new_ca* and *new_ua* respectively[4].

Step 2 (Forward chaining): For all the conditional answers in *old_ca* and *new_ca*, resolve the first goal in the body (which is an instance of Q) with all the answers from *new_ua*. Do the same for all the conditional answers in *new_ca* with all the unconditional answers from *old_ua*. Put all the resolved clauses in *to_unfold*, and add *new_ca* and *new_ua* to *old_ca* and *old_ua* respectively. If after this $to\_unfold = \{\}$, then stop. Otherwise, go back to step 1.

The answers are stored in the list *old_ua*. Since the algorithm might not terminate (there can be infinitely many answers), we display the answers to the user when they are added to *old_ua*.

4.3. Examples

In this section we will show two examples of how the system works. The first example shows how all the answers are collected during the unfolding step. Consider the following program:

```
p(X) ← q(X);
p(a);
p(b);
q(X) ← p(X);
q(c);
```

We will show how the algorithm finds all answers to the query ?p(X), by showing the content for each of *old_ca*, *old_ua*, *new_ca*, *new_ua* and *to_unfold* after each step of the algorithm.

Step 0:
$old\_ca = \{\}$
$old\_ua = \{\}$
$new\_ca = \{\}$
$new\_ua = \{\}$

[4] Throughout this algorithm we check if an answer (both conditional and unconditional) is an instance of an already existing answer before we add it to any list of answers. This is to avoid duplication of answers.

to_unfold = {p(X) ← q(X); p(a); p(b)}

Step 1:
old_ca = {}
old_ua = {}
new_ca = {p(X) ← p(X)}
new_ua = {p(a); p(b); p(c)}
to_unfold = {}

Step 2:
old_ca = {p(X) ← p(X)}
old_ua = {p(a); p(b); p(c)}
new_ca = {}
new_ua = {}
to_unfold = {}

after which the algorithm terminates. During step 2 of the algorithm no new clauses are generated (if we would not check if clauses are instances of already existing answers this would not be the case), and thus the algorithm stops.

The second example illustrates the forward chaining step. We will use the algorithm to find all solutions to the query ?t(X,Y) using the following program:

t(a,b);
t(b,c);
t(X,Z) ← t(X,Y) t(Y,Z);

Since the unfolding step never unfolds anything (all the clauses in *to_unfold* either are facts or the first goal in the body is an instance of t(X,Y), and thus during step 1 the facts are simply put into *new_ua*, and the rules into *new_ca*), we only show the content of *old_ca*, *old_ua* and *to_unfold* after step 2 (both *new_ca* and *new_ua* are empty at this point) is performed.

Step 0:
old_ca = {}
old_ua = {}
new_ca = {}
new_ua = {}
to_unfold = {t(a,b); t(b,c); t(X,Z) ← t(X,Y) t(Y,Z)}

Step 2:
old_ca = {t(X,Z) ← t(X,Y) t(Y,Z)}
old_ua = {t(a,b); t(b,c)}
to_unfold = {t(a,Z) ← t(b,Z); t(b,Z) ← t(c,Z)}

Step 2:
old_ca = {t(X,Z) ← t(X,Y) t(Y,Z); t(a,Z) ← t(b,Z); t(b,Z) ← t(c,Z)}
old_ua = {t(a,b); t(b,c)}
to_unfold = {t(a,c)}

Step 2:
old_ca = {t(X,Z) ← t(X,Y) t(Y,Z); t(a,Z) ← t(b,Z); t(b,Z) ← t(c,Z)}
old_ua = {t(a,b); t(b,c); t(a,c)}
to_unfold = {t(a,Z) ← t(c,Z)}

Step 2:
old_ca = {t(X,Z) ← t(X,Y) t(Y,Z); t(a,Z) ← t(b,Z); t(b,Z) ← t(c,Z); t(a,Z) ← t(c,Z)}
old_ua = {t(a,b); t(b,c); t(a,c)}

$to\_unfold = \{\}$

after which the algorithm stops.

5. Properties of Algorithm 1

In this chapter we will prove that the previously discussed algorithm has the following properties:

(1) Correctness: every answer produced by the algorithm is a logical consequence of the original program.

(2) Completeness, in the sense that:
a) All the answers (logical consequences) of the original program are also logical consequences of a transformed program (we will define what we mean by "transformed program").
b) If the algorithm terminates, we know it has produced all possible answers for the query.

In fact, all answers to the query will be produced. We do not guarantee that the algorithm eventually stops, even if there are only finitely many answers. At the end of this chapter we discuss minor modifications that can be made to the algorithm without changing the correctness and completeness aspects of it.

Before we prove these results we should first note to which type of Prolog programs the algorithm is restricted. It can handle all Pure Prolog (for terminology see [Sterling86]) programs, together with the arithmetic and type-checking predicates. It also handles the meta-logical predicates correctly (is_var, var, ...), but cannot deal with extra-logical predicates (I/O predicates, and those modifying the set of clauses) and the cut predicate. Negation is currently handled by Prolog itself, and thus operates correctly only if Prolog itself can handle it correctly (as far as infinite loops go).

5.1. Correctness

The following theorem proves the correctness (soundness) of our algorithm:

Theorem 5.1: Let P be a program, and Q a query. Then every answer produced by our algorithm is a logical consequence of the program.

Proof: We only use SLD-resolution to find the answers to a query (note that when we "forward chain" using the unconditional answers, we are merely applying unit resolution, which is a special case of SLD-resolution), and by the soundness of SLD-resolution (see [Lloyd84]) we can conclude that any answer computed by our algorithm must thus be a logical consequence of the program. ■

5.2. Completeness

Before we can prove the completeness results, we first have to define what we mean by a transformed program for our algorithm. In order to do that, we will first show that step 1 of the algorithm always terminates.

Lemma 5.1: For any conditional answer C generated by the algorithm, the unfolding of step 1 of the algorithm will always terminate.

Proof: Assume this is not the case, and let $Q\theta$ be the head of the clause for which the unfolding does not terminate. The only way in which this can happen is that we are on some infinite branch of the search tree which is rooted at $Q\theta$. However, since all conditional answers have been obtained using SLD resolution, this branch is also part of an infinite branch of the original search tree. Therefore, by property (2), there must be a goal appearing on this branch which is an instance of Q. But this contradicts the fact that this step does not terminate. ■

Clearly step 2 of the algorithm also terminates. And thus given a program P and a query Q, we define the **transformed program** after step 1 of the algorithm to be the program P, with those clauses whose heads (we consider a fact as a clause consisting of a head and an empty body) have the same predicate symbol as the query Q, replaced by all the unconditional and conditional answers in *new_ca*, *new_ca*, *old_ca* and *old_ua*. We thus define a new transformed program each time step 1 of the algorithm is executed. If the algorithm terminates there is a unique final transformed program.

In the proof of completeness we will use the following result of Tamaki and Sato ([Tamaki83]):

Lemma 5.2: Let C be a clause in program P, and G an arbitrary goal in the body of C. Then the program P', obtained from P by replacing C by the clauses resulting from unfolding G, is equivalent to the original program P.

The following theorem proves the first part of our completeness result.

Theorem 5.2: Let P be a program, and Q a query. Then every transformed program P' produced by our algorithm is such that any ground instance[5] of Q which can be proven using P can also be proven from program P'.

Proof: We first prove that the program P_0, which is obtained from P by replacing those clauses with the same predicate symbol as Q by all clauses from *to_unfold* after step 0 of the algorithm, has this property. We prove this by splitting up these clauses in three cases, and showing that whenever we use a clause in P, there is a corresponding clause which can be used in P_0.
1) The head of the clause does not unify with Q. But then by property (3) we know this clause can never be used in any proof tree rooted at Q of program P.
2) The head of the clause is an instance of Q. Then the clause will be in *to_unfold*, and thus can be used in P' in exactly the same way as in P.
3) The head H unifies with Q (but is not an instance of Q) with most general unifier θ. In this case the clause C will be replaced by the clause $C\theta$. But again, by property (3) we know that any proof of a ground instance of Q in P using this clause C, uses an instance of this clause such that the head of it is an instance of Q, say $Q\sigma$. But we know that $H\theta$ is the same as Q, and thus $Q\sigma$ must be the same as $H\theta\sigma$, which means that we can use the instance $C\theta\sigma$ of the new clause $C\theta$ to give the same proof in P_0.

We now prove that replacing the clauses from *to_unfold* by all the frontiers of answers, does not eliminate any answers. The way in which we calculate the frontier is exactly by using the unfolding rule repetitively, and thus by Lemma 5.2 we know that this is the case (here it is important that we include all the answers for all the frontiers).

Since during step 2 we only add new clauses to the program, we clearly cannot eliminate any answers during this step. Thus combining the fact that after step 0 we have an equivalent program, and that we do not lose any answers during steps 1 and 2, gives us a proof by induction that each of the transformed programs has the desired property. ■

The final theorem we will prove, shows that if the algorithm terminates it has produced all the answers to the query.

Theorem 5.3: Let P be a program, and Q a query. If the algorithm terminates it has produced all answers that are a logical consequence of P.

Proof: Assume this is not the case, and let $Q\theta$ be a ground instance of Q which can be proven from P, but which was not given as an answer by our algorithm. By Theorem 5.2 we know this goal can also be proven from the final transformed program P' of our algorithm. Let T be a proof tree of $Q\theta$ in P'. We will prove that $Q\theta$ can be proven in P' using a single fact, which means it is an instance of one of the unconditional answers produced by our algorithm. We prove this by showing that if the number of clauses (facts and rules) from P', whose heads are instances of Q, used in T is greater than one, we can reduce that number by one.

So suppose this is the case, then we also know that we must have used at least one rule whose head is an instance of Q from P' in T (if no such rule was used then $Q\theta$ must have been proven by a fact, which would finish our proof). Moreover, we must have used such a rule somewhere in T such that the leftmost goal in the body is proven using a fact from P' (this is because all the rules whose heads are instances of Q in P' have as their leftmost goal in the body another instance of Q). Let

[5] The meaning of a program (see [Tamaki83]) is defined to be the set of all ground instances which can be proven from it, for that reason we are only interested in those answers which are ground instances. Answers with variables appearing in them are understood to be universally quantified. This means we consider an answer proven if it is an instance of one of the answers produced by our algorithm.

$R = (Q\theta_1 \leftarrow G_1 \cdots G_n)$ be such a rule, which was used to prove goal $Q\theta_0$ (this is not necessarily the same goal as $Q\theta$) in T with most general unifier σ_1. Now G_1 is an instance $Q\theta_2$ of Q, and suppose $Q\theta_2\sigma_1$ was proven using fact $F=Q\theta_3$ from P' with most general unifier σ_2.

During the algorithm we resolved G_1 in R using F, and we also know this resolution must have succeeded since these two goals unify ($G_1\sigma_1\sigma_2=F\sigma_2$), say with mgu σ_3. If we now look at the path in the frontier in which the same or-choices are made as in T (since σ_3 is a most general unifier, part of the subtree of T below $Q\theta_0\sigma_1$, which is the same as $Q\theta_1\sigma_1$, is an instance of this part of the frontier rooted at $Q\theta_1$), there are two possibilities:
(1) In calculating the frontier we never encounter an instance of Q. But then P' contains an unconditional answer of which $Q\theta_0$ must be an instance. We can then use this single fact rather than R and F to prove $Q\theta_0$ in T.
(2) In calculating the frontier we encounter the instance $Q\theta_4$ of Q. In this case P' contains a conditional answer of the form $Q\theta_1\theta_5 \leftarrow Q\theta_4\ G`_1 \cdots G`_n$ (where θ_5 is the composition of all the substitutions that were made in the calculation of this conditional answer) such that $Q\theta_0$ is an instance of $Q\theta_1\theta_5$. And again, we can simply use this instance of this rule rather than using fact F and rule R.

In each case we show that we can reduce the number of clauses whose heads are instances of Q by one, which completes our proof. ■

The algorithm will also eventually produce all the answers to a query. This is because we know that when the algorithm terminates it has produced all answers. So unless all answers are produced the algorithm will repeat steps 1 and 2 (both steps are always guaranteed to terminate) infinitely often. And by a similar inductive argument as used in theorem 5.3 we can show that each answer must be produced after a finite number of repetitions of steps 1 and 2.

5.3. Modifications to the Algorithm

In this section we suggest three minor modifications which can be made to the algorithm and its conditions, without changing the correctness and completeness properties.

First of all it should be noted that condition (3) is actually not needed. Whenever a goal Q' appears in the derivation which is not an instance of the initial query Q, but has the same predicate symbol, we have to make sure we unfold it using the clauses of the original program P. This is because we cannot only use those clauses whose heads unify with Q, since we could then eliminate some of the answers for this goal Q'. This is in fact the version of the algorithm that is implemented. The reason for mentioning this extra condition is that the definition of **transformed program**, and hence also the proofs using this definition become a lot simpler this way. To give a formal proof without the third condition, we could copy all the clauses with the predicate symbol of Q using a new (i.e. not appearing elsewhere in the program) predicate symbol, and for all the goals that are more general than Q and that appear in the body of some rule we could replace that goal by a corresponding goal using this new predicate symbol.

Currently the algorithm checks whenever a conditional or unconditional answer is added to any of the lists of answers, that it is not an instance of an answer already in one of those lists. This is to avoid duplication of answers, and in fact is not needed to guarantee correct behavior. The reason we perform this check, is that we want to avoid cases where we generate the same answer infinitely often, which would happen for the following program:

p(a);
p(a) ← p(X);

If we would now ask the query ?p(X) and the check for instances of answers would not be included, we would generate the answer p(a) infinitely often.

A further improvement to the algorithm would be the elimination of clauses that we know can only generate answers that can also be generated by the other clauses of the program. This is the case when one of the answers (conditional or unconditional) is a logical consequence of the others. Trivial examples of this are:

- The answer is an instance of one of the already existing ones (in fact we use this simplification in our algorithm).
- The answer is a tautology. An example of this is when a goal appears in the body of a rule which is identical in binding to the head of the rule (a different proof of why we can eliminate a derivation in which this happens is given in [Poole85]).

6. More General Loop Checking Algorithm

We will give an outline of an algorithm which would work correctly for all programs for which only condition (1) is true. We want the algorithm to have the following properties:

- It must produce only correct answers, and no possible answers should be eliminated.
- It should use the standard Prolog inference rule whenever possible.
- It should avoid infinite branches for all programs satisfying condition (1).

Besides these properties we of course want the algorithm to be as efficient as possible. Thus given a program P and a query Q, we suggest the following outline for an algorithm:

Step 1: Starting with the goal Q, unfold the goals using the standard Prolog inference rule, while keeping track of all the goals resolved along a branch[6]. Whenever a goal is an instance of a goal higher up the branch, do not unfold that goal any further, and continue unfolding the other goals in the body[7]. Repeat this process until there are no goals left or all the goals in the body are instances of goals higher up the branch. At the end collect all the conditional and unconditional answers for the original query, as well as all the conditional and unconditional answers for those goals which have an instance of them appearing on a branch below them.

Step 2: Use forward chaining for the set of all remaining goals to find answers to the original query.

The main difference with the original algorithm is that all the unfolding is done during step 1 of this algorithm. All the goals appearing in the bodies of the conditional answers are thus instances of goals appearing somewhere above them in the search tree. This means they can be solved by using a forward chaining strategy. As before, we can guarantee that step 1 of the algorithm terminates, and that the transformed program is equivalent to the original program.

The reason for not specifying the second step of this algorithm, is that there are many different strategies for performing this step. The main problem here is that somehow we want a systematic way of finding new answers to the original query using forward chaining. From step 1 of the algorithm we get a set of conditional and unconditional answers. The bodies of the conditional answers consist of goals that are instances of the goal for which the particular conditional answer was created, or that are instances of other goals for which we also have a set of conditional and unconditional answers. If we somehow have an infinite number of answers for one of the goals in the body of a rule, only one of which leads to an actual answer for that rule, there is the danger of never generating an answer at all using that rule if the algorithm is not guaranteed to generate all answers for that particular goal.

Although it seems that we are back at square one (our algorithm goes into an infinite loop without generating any answers while there are still more answers to the query), we have in fact changed the type of problem we have to deal with. Before, the Prolog machine would go down an infinite branch without ever proving anything. Now, it performs useful work in that it proves (i.e it actually finishes the proofs) an infinite number of subgoals. Moreover, we believe that it is possible to apply forward chaining in such a way that we can guarantee that all answers will be found (as is the case for the first algorithm).

---

[6] Since we are keeping track of all the goals higher up the branch, this algorithm would work equally well if the query would contain more than one goal.

[7] This means that we execute the goals in a different order than Prolog, which means we cannot guarantee correct behavior for programs with builtin predicates which depend on certain variables being instantiated (such as the arithmetic and type-checking predicates).

This algorithm is necessarily also a lot less efficient than the first one. First of all, the number of instance checks that has to be performed for algorithm 1 is linear in the length of the path (one check for each goal on the path), whereas for this algorithm it is quadratic in the length of the path (have to compare each goal with all the goals appearing above it). Similarly in space; for algorithm 1 we only need to save all answers for the original query, whereas for this algorithm we need to keep track of all the solutions for all the queries in a subtree.

Currently, condition (1) is based on the search tree of the query. One type of program for which this condition is guaranteed to hold is if there are no functor symbols (including lists) appearing in the program. This means that every term appearing in the search space of the query is either a variable, or one of the finite number of constants appearing in the program and the query. And thus we can only have a finite number of goals that are not variants of each other. This in turn means that every infinite branch of the search tree contains a goal which is an instance of a goal higher up the tree.

7. Future Work

As only an outline of the second algorithm was given, the first area which should be mentioned for future work is the implementation of such an algorithm. The implementation of the unfolding part of the algorithm is straightforward (although we expect it to be very inefficient). A possible implementation of the forward chaining would be more complicated. An important point to consider there is to somehow guarantee that all answers are eventually generated.

Other improvements to the algorithm can be made by making it handle negation better, or by applying the algorithm only for certain predefined predicates. The latter seems to be a very promising application since most relations defined in a program do not have a problem with looping, while others can be more easily defined if we somehow had a loop check available (e.g. transitive and symmetrical relations). This would also reduce the overhead (both time and space) of loop checking considerably.

After having obtained a satisfactory loop checking algorithm for Prolog, we can see if it can be applied to more general reasoning systems which are based on Prolog (such as Theorist [Poole86]). Due to the inherent inefficiency of the more general loop checking algorithm we would probably want a version of it which only checks for loops for certain predefined predicates.

A final area of work would be to give a more exact definition of what type of program can now be dealt with. Currently, the condition which must hold for the algorithm to behave correctly is based on the search tree of the query. We already showed that this condition always holds if there are no functors in the program, it would be nice if we could could give other (more general) cases for which this condition must hold.

8. Conclusion

We have successfully implemented an initial version of a Prolog interpreter with loop checking. Although the correct behavior of this algorithm is only guaranteed if certain conditions are met, this algorithm is powerful enough to deal with all the examples discussed in earlier papers on this subject ([Covington85a], [Covington85b], [Nute85] and [Poole85]).

A more general algorithm is suggested which would work for the most general class of programs we can reasonably deal with, without expecting an enormous decrease in efficiency. An implementation of this problem could suffer from seemingly similar problems as those we were trying to avoid. But as we pointed out, even if this is the case we would still have achieved a considerable improvement over the standard Prolog machine.

9. References

[Brough84] D.R. Brough, A. Walker, "Some Practical Properties of Logic Programming Interpreters", *Proceedings of the International Conference on Fifth Generation Computer Systems*, 149-156.

[Burstall75] R.M. Burstall, J. Darlington, "A Transformation for Developing Recursive Programs", *Journal of the ACM*, vol.24, no. 1, 44-67.

[Chang73] C. Chang, R.C. Lee, "Symbolic Logic and Mechanical Theorem Proving", Academic Press, New York, 1973.

[Cheng87] M.H.M. Cheng, M.H. van Emden, P.A. Strooper, "Logic-based Program Transformation", in preparation.

[Covington85a] M.A. Covington, "Eliminating Unwanted Loops in Prolog", *ACM SIGPLAN Notices*, vol. 20, no.1, 20-26.

[Covington85b] M.A. Covington, "A Further Note on Looping in Prolog", *ACM SIGPLAN Notices*, vol. 20, no.8, 28-31.

[Kowalski79] R. Kowalski, "Logic for Problem Solving", Elsevier North Holland, New York, 1979.

[Lewis81] H.R. Lewis, C.H. Papadimitriou, "Elements of the Theory of Computation", Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[Lloyd84] J.W. Lloyd, "Foundations of Logic Programming", Springer-Verlag, Berlin, 1984.

[Nute85] D. Nute, "A Programming Solution to certain Problems with Loops in Prolog", *ACM SIGPLAN Notices*, vol. 20, no.8, 32-37.

[Poole85] D. Poole, R. Goebel, "On Eliminating Loops in Prolog", *ACM SIGPLAN Notices*, vol. 20, no.8, 38-40.

[Poole86] D. Poole, R. Goebel, R. Aleliunas, "Theorist: a Logical Reasoning System for Defaults and Diagnosis", Research Report CS-86-06, University of Waterloo, Februari 1986.

[Sterling86] L. Sterling, E. Shapiro, "The Art of Prolog", MIT Press, Cambridge, Massachusetts, 1986.

[Tamaki83] H. Tamaki, T. Sato, "A Transformation System for Developing Recursive Programs", ICOT Technical Report TR-018, August 1983.

Combining Dependency Directed Backtracking with the ATMS

Bruce Spencer
CS786

April 25, 1987

1 Introduction

Problem solvers can benefit from the techniques of dependency directed backtracking and the ATMS[4]. In this paper we propose a method for implementing them in the same system. The reasons for combining them are simple. First, dependency directed backtracking needs the information that truth maintenance provides. Second, dependency directed backtracking keeps the additions to the truth maintenance system to a minimum. Third, if an inconsistency forced an inference to be undone, but the eventual proof requires that inference, the truth maintenance system will keep it in the background so it does not have to be redone. These three reasons can be summarized: we can identify the points where inconsistencies arise, make new inferences only at those points, and never make the same inference twice.

The paper will first describe some basic concepts. Then two proof procedures will be presented, one using the ATMS and another combining the ATMS with dependency directed backtracking. Each is followed by an example. We conclude with some brief remarks and suggestions for future work.

2 Basic Concepts

2.1 Horn Clause Theorem Proving

A logic program is a set of Horn clauses, each composed of a head and a body of zero or more elements. The head and body elements are predicates, which are constructed from a predicate symbol (a string of characters) and zero or more terms following it, surrounded by parentheses. Terms are either logical variables, or constructions of a functor (also a string of characters) followed by zero or more terms surrounded by parentheses.

A logical variable can be substituted by any term. All variables with the same name in a clause are simultaneously substituted with the same term. Unifying two predicates means finding a set of substitutions for the variables which makes the predicates lexically identical. The most general unifier is the unique set of substitutions that is implied by the equality of the terms.

The logic program, in combination with a proof procedure can be used decide if a *query*, a list of predicates, has a proof. The proof procedure is said to prove the query if it can find for each *goal*, a predicate in the query, one clause from the program whose head unifies with the goal and whose body, with the substitutions in effect, can also be proved. All proof procedures use a tree, called a proof tree, to represent the clauses used to prove the query.

The proof procedure must have a mechanism for deciding what predicate to prove next. For example, Prolog uses a left to right seach across goals. It searches depth-first; it proves the body of the clause chosen for a goal before moving right to another goal.

A proof procedure must also deal with different substitutions for the same variable arising from different branches in the proof tree. Most proof procedures use backtracking, the subject of the

next section.

2.2 Backtracking

When building the proof tree, choosing clauses for a goal is a nondeterministic operation. Often only a small subset of the clauses available for a goal will lead to a proof. For example if a pair of goals $f(X)$ $h(X)$ has one variable in common X, and there are m clauses for f while there are p clauses for h there are $m \times p$ combinations that in the worst case must be checked to find a substitution for X.

Most proof procedures backtrack, which means they choose one clause for each goal, and throw away the clause for one of the goals if an inconsistency is discovered, such as substituting two different terms for X. Thus it must prove $m \times (p + 1)$ clauses to check the $m \times p$ combinations.

Prolog does chronological backtracking, where the backtrack points are chosen in reverse order of the left to right, depth first search. In the example $f(X)$ $g(Y)$ $h(X)$ where there are m clauses for f, n clauses for g and p clauses for h, Prolog would, in the worst case, prove $m \times (1 + n \times (1 + p))$ clauses. Clearly the work done reproving g is wasted since it cannot affect the substitutions for X.

(In fairness to Prolog, an optimization makes the above example require as few as $m \times (1 + 2n)$ proofs. The substitution for X made by f is carried over when we try to prove h so h may require only one step. But a translation of the problem to $f(X)$ $g(Y)$ $h(X1)$ $eq(X\ X1)$ will prevent this optimization from applying. The only clause for eq is $eq(XX)$.)

2.3 ATMS

Truth maintenance systems maintain the support for each piece of data, so that it need not be rederived. The ATMS maintains the support in terms of the underlying assumptions. (When we are building proof trees, an assumption corresponds to a choice of a clause for a goal.) Each piece of data is supported by one (or more) conjunctions of assumptions and other data. The other data is itself supported by further data and assumptions. Eventually the support must be in terms of assumptions. Thus the support for a datum may be expressed as an and/or formula of assumptions which is isomorphic to the tree of justifications. With each datum is stored its *label*, a concise description of its support expressed in disjunctive normal form (DNF), a disjunction of conjunctions of assumptions. This label is a translation of the and/or formula into DNF, but reduced in two ways. Any conjunction of assumptions, called and *environment*, is removed from the label if it is a superset or duplicate of another environment. Also any environment which is a nogood or a superset of a nogood is removed. The label is sound and complete because it is logically equivalent to the supporting justifications. It is also minimal since duplicates and supersets are removed. Finally it is consistent since inconsistent environments are removed.

In the ATMS, one datum, $\perp$, corresponds to inconsistency. When a set of data is found to be incompatable, we can record this by justifying $\perp$ with the data. Thus the label of $\perp$ would be the set of minimal nogoods. But since nogoods are not part of labels, they are not stored with $\perp$, but in a separate nogood relation.

Using only truth maintenance techniques we can build an interpreter which does no backtracking and finds all solutions. This interpreter would reduce the number of proofs in the above example to $m + n + p$.

2.4 Backtracking as diagnosis

Dependency directed bactracking means finding a goal which contributes to the inconsistency and reproving it. The dependency directed backtracking algorithm described later is based on the diagnosic procedure described in deKleer[6]. There deKleer defines a *candidate* as a set of assumptions, all of which may be inconsistent according to the information available when it is generated. It is generated from the nogoods by constructing a set of assumptions chosen one from each nogood, and repeating this for all combinations. For example, if inconsistent variable substitutions imply that the set of nogoods is $(a1, a2)$ and $(a2, a3, a4)$ then the candidates would be $(a1, a2), (a1, a3), (a1, a4), (a2), (a2, a3), (a2, a4)$.

Any candidate which is a superset of another candidate is rejected. We are left with $(a2), (a1, a3), (a1, a4)$.

Intuitively, one of the candidates generated in this way will contain only false assumptions, but we can not tell which one. This candidate contains exactly the assumptions we need to undo to remove the inconsistency. We choose one candidate and reprove the goals associated with those assumptions.

Using dependency directed bactracking in the previous example, we would do $m \times p + 1$ proofs to find a substitution for X. We shall see that with truth maintenance techniques and dependency directed backtracking, we will reduce the number of proofs to $m + 1 + p$. This reduction comes as a result of an increase in the cost of each proof. Label updating, conflict recognition and conflict resolution are procedures which Prolog does not require.

3 Supporting Routines

In the next section, we will be discussing proof procedures and their implementations. This section outlines the support routines available to proof procedures.

Choose_goal takes a list of goals to be proved and selects one, perhaps heuristically, and returns that goal and the smaller list. A left to right search is done if the first goal is always selected.

A routine, *get_clause*, gets a clause that unifies with a goal. It may use a heuristic. It will never return the same clause twice, and must eventually return all clauses for subsequent calls. It renames the variables to unique names, that will not be reused while this top level goal is being proved. These names are of the form $\_n$ where n is an integer. Another routine is available to say when no clauses remain.

Another routine, *new_assn* creates a new ATMS assumption. Its name is also unique for the duration of this proof. The names are of the form a_n where n is an integer. It generates the integers in increasing order.

The ATMS routine, *add_just* is provided by the ATMS to allow justifications to be added to data. Either predicates or variable substitutions are justified. e.g. $add\_just((\_i := fred) < -a5)$. We refer the interested reader to [4] for more details on this routine.

Another ATMS routine *believe* takes an environment and a datum and says whether this datum's support exists in this environment. It is possible to use believe to find out a variable's substitution given an environment.

Another routine, *unify*, is available to unify the head of a clause with a goal. In practice, this would be implemented together with get_clause, since they do the same work. It captures the variable substitutions in a list of the form $(\_i := term)$, which means this variable $\_i$ is substituted

by the term *term*. This list is split into two lists, one list of substitutions for the head and one for the goal. There will never be a variable which occurs in both since get_clause generates unique names. The substitution of a variable with another variable is considered a substitution for the head. The substitutions for the head will be applied to both the head and the body of the clause, which means we need the routine *apply_sub* to do this.

The routine *check_conflict* is called after the body of a clause has been proved. It inspects any variables in the body that are substitued with different terms. If the terms are unifiable, new substitutions can be added. For example, if X is substituted with $w(3)$ as a result of proving one goal and to $w(Y)$ by another, then $(Y := 3) < -(X := w(3))\ (X := w(Y))$ can be added to the justifications. If the terms are not unifiable, an inconsistency must be added to the ATMS. For example, if the two substitutions $(\_13 := 3)$ and $(\_13 := 4)$ exist, then we must add the justification $\perp < -(\_13 := 3)\ (\_13 := 4)$.

Also, any substitutions to variables in the head must be propogated upward to other substitutions where those variables occur in the term. The routine *propogate_sub* takes a substitution and progates it up the proof tree. For example, if the ATMS has that $(\_20 := fred(\_21))$ and we have just found that $(\_21 := 3)$ then we need to add the justification $(\_20 := fred(3)) < -(\_20 := fred(\_21))\ (\_21 := 3)$.

Finally the routine *evaluate(Predicate Env)* applies substitutions to variables in the *Predicate* that have substitutions in the environment *Env* until no more substitutions can be done. It is used to report answers at the end of the proof.

4 Strategies and Algorithms

Two algorithms will be presented. Both are depth first. The first uses the ATMS but not dependency directed backtracking; the second uses both.

4.1 No dependency directed backtracking

A straightforward way to use the ATMS to do theorem proving is to prove all clauses for each goal, and let the ATMS worry about keeping track of the consistent environments. This produces all solutions and is feasible only if no branch in the proof tree is infinite. Otherwise the procedure will not return any answers, even though a left-to-right search might.

Given a query, we add the clause ? < $-$*query* and call prove_one_goal(?). When the procedure returns, we need to evaluate the query in each of the environments of the label of ?. There are as many solutions as there are environments.

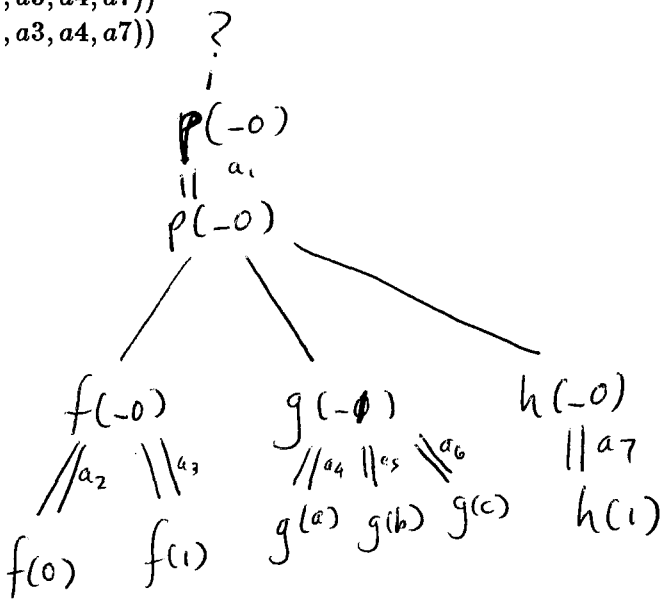
```
prove_one_goal(Goal):
      while there is a clause whose Head unifies with Goal do
            get_clause(Goal Head Body)
            unify(Goal Head Sub_for_Goal Sub_for_Head)
            apply_sub(Sub_for_Head Head)
            apply_sub(Sub_for_Head Body)
            new_assn(Assn)
            add_just(Goal < − Assn Head)
            add_just(Head < − Body)
            add_just(Sub < − Assn) for each Sub in Sub_for Goal
            propogate_sub(Sub) for each Sub in Sub_for_Goal
            prove_goals(Body)
            check_conflict(Body)
      endwhile

prove_goals(Goals):
```

```
if Goals is not empty then
        choose_goal(Goal Goals LeftOvers)
        prove_one_goal(Goal)
        prove_goals(LeftOvers)
endif
```

This example is taken from the discussion in Section 2. It shows the labels after this proof procedure is run. Since there are three environments in the label for ?, there are three solutions. All solutions are identical, p(1).

Program:
$p(X) < -f(X)\ g(Y)\ h(X)$;
$f(0)$;
$f(1)$;
$g(a)$;
$g(b)$;
$g(c)$;
$h(1)$;

Datum	Label
?	$((a1, a3, a6, a7), (a1, a3, a5, a7), (a1, a3, a4, a7))$
$p(\_0)$	$((a1, a3, a6, a7), (a1, a3, a5, a7), (a1, a3, a4, a7))$
$f(\_0)$	$((a3), (a2))$
$g(\_1)$	$((a6), (a5), (a4))$
$h(\_0)$	$((a7))$
$(\_1 := a)$	$((a4))$
$(\_1 := b)$	$((a5))$
$(\_1 := c)$	$((a6))$
$(\_0 := 0)$	$((a2))$
$(\_0 := 1)$	$((a7), (a3))$



Justifications:	Variable Bindings:
$? < -p(\_0)$;	
$p(\_0) < -a1\ p(\_0)$;	
$p(\_0) < -f(\_0)\ g(\_1)\ h(\_0)$;	
$f(\_0) < -a2\ f(0)$;	$(\_0 := 0) < -a2$;
$f(\_0) < -a3\ f(1)$;	$(\_0 := 1) < -a3$;
$g(\_1) < -a4\ g(a)$;	$(\_1 := a) < -a4$;
$g(\_1) < -a5\ g(b)$;	$(\_1 := b) < -a5$;
$g(\_1) < -a6\ g(c)$;	$(\_1 := c) < -a6$;
$h(\_0) < -a7\ h(1)$;	$(\_0 := 1) < -a7$;
$\perp < -(\_0 := 0)\ (\_0 := 1)$;	

Nogoods: $(a2, a3), (a2, a7)$

4.2 A dependency directed backtracking algorithm

Another approach is to use the ATMS to guide dependency directed backtracking. The backtrack candidates are generated from the nogood relation, one is chosen, and this leads to the set of assumptions which must be revoked. When an assumption is revoked, the associated clause is removed from its goal, and another clause is chosen.

The candidates are calculated from the nogoods in the same way that a new label is generated from a (single) justification. The labels in the justification form a conjunction of disjunctions and we want the corresponding disjunction of conjuncts. Similarly, the nogood set is a conjunction of disjunctions of assumptions, and we want to calculate the corresponding disjunction of conjuncts. The major difference is that we consider the label's assumptions to be true so we remove nogoods. With the candidates, we consider the assumptions to be false. We can also define *goods*, and remove them from the candidates.

A good set a set of assumptions of which one must be true. When a goal has used its last clause, an environment can be added to the good set, the set of assumptions associated with clauses used for this goal. If the goal is true then one of the assumptions must be true. When we calculate candidates, we are looking for sets which are all false, so we can remove any good sets or supersets of good sets.

The term *father* here refers to a goal, from the point of view of a predicate in the body of a clause whose head is unified with the goal. The father's *sons* are these predicates in the body. A *right_son* is a son which has not been visited, and a *left_son* is one that has had at least one clause unified to it.

For each goal, we will need the current environment, which is the set of most recent assumptions

that justify the goal. The current environment may or may not be consistent. We will use the current environment to find the point in the tree where an assumption has to be revoked.

A goal is inconsistent if its label is empty.

To prove a query, we start the prove_one_goal, with the current goal set to ?. The only clause for ? is the singleton $? < -\text{query}$.

```
global varaibles: Current_goal, List_of_goals;

prove_one_goal:
    % there is an inconsistency in one of the ancestors of the current goal
    if there are no clauses left for Current_goal
        add the assumptions used in previous clauses for this goal to the good set
        Current_goal := father of Current_goal
        go to conflict_resolution
    else
        get_clause(Current_goal Head Body)
        unify(Current_goal Head Sub_for_goal Sub_for_head)
        apply_sub(Sub_for_head Head)
        apply_sub(Sub_for_head Body)
        create_assn(Assn)
        add_just(Current_goal < - Assn Head)
        add just(Head < - Body)
        add_just(Sub < - Assn) for each Sub in Sub_for_goal
        propogate_sub(sub) for each Sub in Sub_for_goal
        List_of_goals := Body
        go to prove_sons

prove_sons:
    % attempts to prove the List_of_goals
    % Father refers to the father of List_of_goals
    if List_of_goals is empty
        check_conflict(left_sons of Father)
        if Father is consistent
            if the Father is ?
                stop with success
            else
                List_of_goals := right_sons of father of Father
                go to prove_sons
        else
            Current_goal := Father
            go to conflict_resolution
    else
        choose_goal(Current_goal, List_of_goals)
        make this goal a left son of Father
        go to prove_one_goal
```

```
conflict_resolution:
    % the current goal is inconsistent
    generate the candidates from the nogoods
    if there is a candidate with a nonempty intersection with the
        Current_goal's current_environment then
        Candidate = this intersection
        while the candidate is not a singleton do
            pick a left son of Current_goal
                whose current_environment has
                a non-empty intersection with the candidate
            Current_goal := this left son
            Candidate = this intersection
        end while
        go to prove_one_goal
    else
        if Curent_goal is ? then
            stop with failure
        else
            Current_goal := father of Current_goal
            go to conflict_resolution
        endif
    endif
```

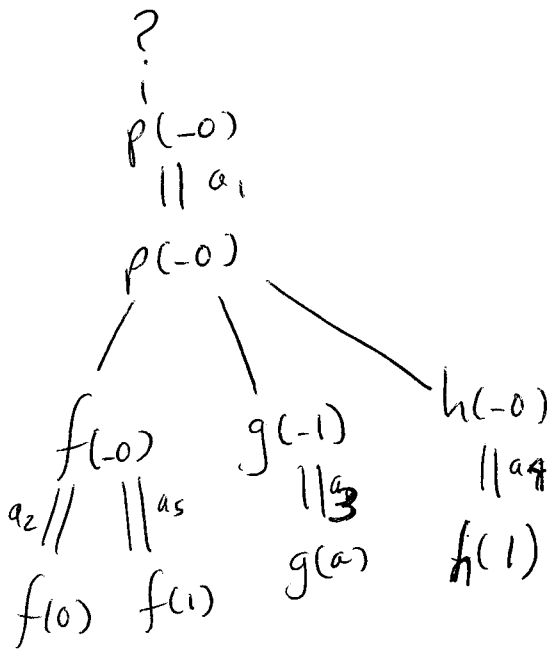
This example uses the same program as the previous example. To prove p, we call prove_one_goal on each of f, g, and h. Then check_conflict notices that _0 is substituted with 0 and 1, which are not unifiable. As a result, the nogood $(a2, a4)$ is produced. The candidates $(a2)$ and $(a4)$ are generated. If the candidate $(a4)$ is chosen, we attempt to reprove h, but there are no clauses for h, so the set of assumptions used for h, the singleton $(a4)$, is added to the good set. Since there is still an inconsistency at p, the set of conflicts is again generated, this time yielding only $(a2)$. The other clause for f is chosen, and since there is no remaining inconsistency, we are done.

Earlier the claim was made that $f(\_0)\ g(\_1)\ h(\_0)$ could be proved in $m + 1 + p$ proofs. In this example we have done it in 4 proofs, where m is 2 and p is 1.

Datum	Label
?	$((a1, a3, a4, a5))$
$p(\_0)$	$((a1, a3, a4, a5))$
$f(\ 0)$	$((a5), (a2))$
$g(\_1)$	$((a3))$
$h(\ 0)$	$((a4))$
$(\_1 := a)$	$((a3))$
$(\ 0 := 0)$	$((a2))$
$(\_0 := 1)$	$((a1, a5), (a1, a4))$

Justifications:	Variable Bindings:
$? < -p(\_0)$;	
$p(\_0) < -a1\ p(\_0)$;	
$p(\_0) < -f(\_0)\ g(\_1)\ h(\_0)$;	
$f(\ 0) < -a2\ f(0)$;	$(\ 0 := 0) < -a2$;
$g(\_1) < -a3\ g(a)$;	$(\_1 := a) < -a3$;
$h(\ 0) < -a4\ h(1)$;	$(\ 0 := 1) < -a4$;
	$\perp < -(\_0 := 0)\ (\_0 := 1)$;
$f(\_0) < -a5\ f(1)$;	$(\_0 := 1) < -a5$;

Nogoods:	Goods:
$(a2, a4)$;	$(a4)$;



4.3 Discussion

This section contains some observations about the algorithms.

We use a number of goto's in the algorithm in section 4.2, which is usually considered bad style. But in this case we are traversing a tree which is changing at internal and external nodes so it is not natural to use recursive routines. It is better to think of the algorithm as a three state machine and the goto's as state transitions. In section 4.1, we are doing updates to only the leaves so a recursive routine is natural.

The conflict resolution phase uses the idea of a singleton candidate. This corresponds to undoing one assumption at a time. If we undo more than one assumption, we may miss solutions. We are guaranteed to find a singleton because each step from father to son eliminates at most one assumption from the candidate.

The conflict resolution phase fixes one conflict at a time. If there is more than one, or if this fix generated further conflicts, we will find them in prove_sons, as we go up the tree looking for inconsistent nodes.

5 Comparison with Other Work

In [7], deKleer combines dependency directed backtracking with the ATMS. The motivation and results are similar to ours. Based on the outline of the algorithm, however, it bears little similarity to ours; it is based on examining environments and scheduling consumers once an environment has been chosen.

As we have already mentioned, the ATMS salvages the work done by unification and propogation. In general backtracking should not be coupled with destroying variable substitutions because the substitutions may become necessary again, in the future. A dependency directed backtracking interpreter for Prolog, described in [2], Cox redoes the work undone by backtracking.

Bruynooghe and Perreira[1] do a subset of dependency directed backtracking. They do not detect all the minimal conflicts. They also lose information about why subtrees in the proof tree were inconsistent. They mention that this was a pragmatic decision.

With the ATMS, we do not lose any information. This may be a problem; there is no way to reclaim the space used by assumptions and variable substitutions even if the problem solver can guarantee that some assumption will never be used again.

In fact it is possible that this approach will never be efficient, or able to deal with large problems. deKleer[4] mentions that his ATMS is capable of dealing with 1000 assumptions. This means the program can only use only 1000 clauses, including those which have been backtracked and are no longer useful.

6 Conclusion

We have seen that truth maintenance without dependency directed backtracking makes sense if all answers to a query are required, and if no branch of the search tree is infinite. Otherwise, it is better to use dependency directed backtracking. It is also clear that dependency directed backtracking without truth maintenance does more work since it must rebuilds structures which were removed by backtracking. In this paper we have explored how we can derive the benefits of both in one algorithm. The feasibility of this approach depends upon the efficiency of the ATMS.

7 Future Work

Future work should include an implementation of the algorithm in section 4.2. (The algorithm in 4.1 has been implemented in Prolog. The dependency directed backtracking algorithm can not be implemented in Prolog in its present form.)

There has been no attempt to share the structures which make up the terms in the substitutions. As variable substitutions are made, the terms are copied and may occur many times in various stages of instantiation. Another approach is to build the structures in an independent space and have the substitions point from the ATMS into this space. Unifications will be represented as arcs between variables and values. These arcs will be tagged with the assumptions that are associated with the unification. Evaluating a variable in a given environment corresponds to following all of the arc that are tagged with assumptions in the environment.

With this approach, the ATMS's substitions contain pointers to the actual structures, so may variables can use the structure without each needing a new copy. But even so there is one ATMS node for each term or instance of a term substituted for a given variable, which may still be too expensive.

It may be possible to use truth maintenance techniques to eliminate entire branches in the proof tree. If two goals are identical, with identical substitutions then it is certainly possible to prove both with the same clause. If two goals only match in name and arity, it may be possible to reuse some of the work done proving one to prove the other.

References

[1] Bruynooghe, M., Pereira, L. M., Deduction Revision by Intelligent Backtracking, *Implementations of Prolog*, ed. J. A. Campbell, Ellis Horwood,(1984) 194-215.

[2] Cox, P. T., Finding Backtrack Points for Intelligent Backtracking, *Implementations of Prolog*, ed. J. A. Campbell, Ellis Horwood, (1984) 216-233.

[3] de Kleer, J., An assumption-based Truth Maintenance System, *Artificial Intelligence* **28** (1986) 127-162.

[4] de Kleer, J., Extending the ATMS, *Artificial Intelligence* **28** (1986) 163-196.

[5] de Kleer, J., Problem solving with the ATMS, *Artificial Intelligence* **28** (1986) 197-224.

[6] de Kleer, J., Reasoning about Multiple Faults, in *Proceedings Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, (1986) 132-139.

[7] de Kleer, J., Back to Backtracking: Controlling the ATMS, in *Proceedings Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, (1986) 910-917.

[8] Poole, David; Goebel, Randy; Representation and Reasoning: A Logical Introduction to Artificial Intelligence, course notes, 1986.

Implementing Fixed Predicates in Theorist

Mark Young
Commonsense Reasoning Project
mayoung@watdragon

April 24, 1987

Abstract

This paper describes an extension to Theorist that borrows the notion of fixed predicates from Circumscription. The system is motivated by examples that Theorist does not currently handle "correctly". A description of the theory behind the system is presented and an implementation is given. The paper ends with a brief comparison of this system to Circumscription.

1 Introduction

An interesting feature of Circumscription [M86,L86] is the ability to designate some relations as *fixed*, meaning that their truth value should not be affected by any assumptions made. It seems that this feature might be useful in Theorist [P86,PGA86].

The introduction of fixed predicates is motivated by examples such as the following.

Example 1 Suppose our only default is *ontable*, and the only other things given are the facts

$$red \Rightarrow \neg ontable$$

and

$$\neg blue \Rightarrow cleartop$$

The intuitive meaning is that we can assume blocks to be on the table unless they are red, and that blocks that are not blue have clear tops.

From the above, Theorist will "deduce" *ontable*. Once the block is assumed to be on the table we can conclude that it is not red. If asked to explain *cleartop* in the above example, Theorist will fail.

There are two problems with Theorist's behaviour in this example. First is that we may not want to conclude anything about the redness of a block. Our intuition is, after all, that the more transient *ontable* relation should be explained in terms of the more permanent *red*. Theorist seems to be doing the reverse.

The second problem is that *cleartop* is, in some sense, more believable than *ontable*. If a block is red then we know that *ontable* is *false*. If it is not red then *ontable* can be *assumed true*. On the other hand, if a block is blue then the value of *cleartop* is *unknown*. If the block is not blue then *cleartop* is *known true*. The fact that Theorist explains the less believable predicate and not the more is counter-intuitive.

Better answers in the two cases would be

$$\textit{ontable} \text{ if } \neg \textit{red}, \text{ by theory } \textit{ontable}$$

and

$$\textit{cleartop} \text{ if } \neg \textit{blue}$$

respectively. By making *red* and *blue* fixed predicates an appropriately extended version of Theorist could find such *conditional* answers.

The introduction of fixed predicates will also allow Theorist to handle problems that it currently cannot. Specifically, problems in which our picture of the world is known to be incomplete. At present, if Theorist cannot prove either ϕ or $\neg\phi$ it will treat both as false. By making ϕ fixed, Theorist will be able to reason in the possible worlds with ϕ true as well as those with ϕ false.

This paper describes an extension to Theorist that uses fixed predicates to generate conditional and unconditional explanations (as defined in [P87]).

2 Formalisms[1]

Let us describe the system we want more formally. We use the standard syntax of the first order predicate calculus, with variables appearing in upper case.

F is a set of closed formulae (called *facts*), which are given as true.

[1]This work was motivated by [P87], and most of this section is taken from that paper.

Δ is a set of formulae from which instances can be used as possible hypotheses.

Φ is a set of predicates (said to be *fixed*) about which implicit assumptions should not be made.

Following are the definitions of explainability used by Theorist and by our modified version.

Definition 1 We say formula g is *explainable* if there is some D, a set of instances of Δ, such that

$$F \cup D \models g$$

$$F \cup D \text{ is consistent}$$

D is said to be the theory that explains g.

Definition 2 g is *conditionally explainable* from F, Δ and Φ, if there is a set D of instances of Δ, and a formula C made of instances of elements of Φ (under conjunction, disjunction, negation), such that

1. $F \wedge C \wedge D \models g$
2. $F \wedge C \wedge D$ is consistent
3. if $F \wedge D \wedge C \models \phi$, where ϕ is a formula made from elements of Φ, then $F \wedge C \models \phi$.

D is said to be the theory that explains g, and C is the condition for D.

Definition 3 g is *unconditionally explainable* from F, Δ, Φ if g is conditionally explainable with theories D_i and corresponding conditions C_i for $i = 1, \ldots, n$, such that $F \models C_1 \vee \cdots \vee C_n$.

The idea of conditional explanation is that if the fixed predicates have values such that C is true, then theory D explains g. If there are theories to explain g regardless of the values of the fixed predicates then g is unconditionally explainable. Examples of how these definitions are used may be found in section 4.

3 Implementation

The implementation requires only two major changes to the Theorist program (as it appears in [PGA86, p 13]). One concerns the proof of fixed predicates and the other checking the consistency of defaults used. The modified code appears in Appendix A.

Since the code is derived from Theorist, we still have the problem that explainability is undecidable. As for Theorist, the correctness of the procedure is assured only when the system halts.

Consistency checking may be carried out incrementally. Conditions and defaults are added one at a time (except where consistency can be proved independently, see below), so one of theses additions may be identified as "the" culprit, in the sense that it was the first to result in inconsistency.

Dealing with fixed predicates is not difficult. If c follows from the facts then it can be handled just as any other predicate. Otherwise, we merely add c to the conditions of the proof—assuming this can be done consistently. All we need check is that $\neg c$ does not follow from the proof so far.

Defaults pose a more difficult problem. In Theorist any proof of $\neg d$ would be sufficient to show default d inconsistent. In our extended version a conditional proof of $\neg d$ need not result in rejection of d. The proof can be fixed merely by adding the negation of the conditions for $\neg d$ to the greater proof.

Consider adding default d to a proof with current theory D and current conditions C. Assume that conditions (2) and (3) of Definition 2 hold. Let C_i be the conditions of the ith proof of d from $F \cup D \cup C$ that do not follow from $F \cup D \cup C$. Only if one of the C_i is empty is d inconsistent with the proof so far. If d is consistent then we have for each i

$$F \wedge D \wedge C \wedge C_i \models \neg d$$

and therefore

$$F \wedge D \wedge C \wedge d \models \neg C_i$$

Thus $\neg C_i$ is consistent with the proof to date and may be added without further checks. Once this has been done for each C_i, d may be added to the proof consistently. Condition (3) also still pertains, since any formula from Φ that follows from $F \wedge (D \wedge d) \wedge C$ has, in the form of the $\neg C_i$, been added to C.

Consider Example 1 above. Asked to explain *ontable*, the system uses the default. It discovers that $\neg$*ontable* is true if *red* is true. This predicate is

fixed and cannot be proven one way or the other, so it assumes the worst—that it is true. Thus *red* is a condition on the proof of $\neg$*ontable*. To allow *ontable* to be true, therefore, it is necessary that *red* be false. Thus $\neg$*red* is made a condition of the proof of *ontable*.

Unconditional explanation is not difficult once conditional has been implemented. First the system derives a conditional explanation for the goal. If the conditions are empty, then the solution has been found. If they are not empty, then temporarily add their negation to the data base and look for an unconditional explanation in the new circumstances.

4 The System in Use

We now provide some examples of the system in use. The conditional theory is returned in the form "if *Conditions* then *Theory*." If *Conditions* is empty then *Theory* always applies, and if *Theory* is empty then *Conditions* is sufficient to prove g.

Unconditional explanations are returned in an "if–then–else" format, meaning that if the conditions under the "if" hold then the theory under the "then" applies, otherwise the theory under the "else" applies.

Example 2 The example in Figure 1 shows how the system handles Example 1. Note that without fixed predicates the system is equivalent to Theorist, and therefore gives the answers that Theorist would give: *ontable* explainable without conditions (forcing *red* to be false), and *cleartop* unexplainable. After making *red* and *blue* fixed the system returns the preferred answers: *ontable* explainable if $\neg$*red* and *cleartop* explainable if $\neg$*blue*.

Example 3 Figure 2 compares conditional and unconditional explanation[2]. In this case *ontable* is explainable by $D = \{$*ontable*$\}$ with $C = \neg$*red*. *onfloor* is explainable by $D = \{$*redonfloor*$\}$ with $C =$ *red*. The relation *down* is unconditionally explainable with $D_1 = \{$*redonfloor*$\}$ and $C_1 =$ *red*, and $D_2 = \{$*ontable*$\}$ and $C_2 = \neg$*red*. The interpretation of the first answer the system returns is: "If the block is red then we can assume *redonfloor*, otherwise we can assume *ontable*. In either case *down* follows."

[2]The commands for these two types of explanation have been kept separate for the time being, though I expect they might be consolidated in the future.

```
: default ontable;
: fact    n(ontable) <- red;
: fact    cleartop <- n(blue);

: explain ontable;
answer : ontable
theory : if
         then    ontable
no (more) answers

: explain cleartop;
no (more) answers

: fixed   red blue;
: explain ontable;
answer : ontable
theory : if      n(red)
         then    ontable
no (more) answers

: explain cleartop;
answer : cleartop
theory : if      n(blue)
         then
no (more) answers
```

Figure 1: Example 1 Revisited

```
: default ontable;
: default redonfloor;
: fact    n(ontable) <- red;
: fact    onfloor <- red redonfloor;
: fact    down <- onfloor;
: fact    down <- ontable;
: fixed   red;

: explain ontable;
answer : ontable
theory : if     n(red)
         then   ontable
no (more) answers

: explain onfloor;
answer : onfloor
theory : if     red
         then   redonfloor
no (more) answers

: unconditional down;
answer : down
theory : if     red
         then   redonfloor
         else   ontable
answer : down
theory : if     n(red)
         then   ontable
         else   redonfloor
no (more) answers
```

Figure 2: Conditional vs Unconditional Explanations

Example 4 Figure 3 shows that explainability is not the same as unconditional explainability. Without fixed predicates, Theorist fails to find an explanation for g, because the only possible explanation (*ontable* and *onfloor*) is inconsistent. Making *red* fixed allows the system, in effect, to reason by cases.

```
: default ontable;
: default onfloor;
: fact    g <- red ontable;
: fact    g <- n(red) onfloor;
: fact    n(ontable) <- onfloor;

: explain g;
no (more) answers

: fixed red;
: unconditional g;
answer : g
theory : if     red
         then   ontable
         else   onfloor
answer : g
theory : if     n(red)
         then   onfloor
         else   ontable
no (more) answers
```

Figure 3: Explainability vs Unconditional Explainability

5 Comparisons with Other Systems

Circumscription is the only other system that makes the distinction between fixed and varying predicates. The treatment is not exactly the same: Circumscription requires that the theory explaining the goal be independent of the fixed predicates, whereas we require only a theory for any value of these predicates.

Consider again Example 4. Circumscribing $\{\neg ontable, \neg onfloor\}$ with *red* fixed will add $ontable \vee onfloor$, but not g. Predicate g does not follow from the facts and defaults we are given, so Circumscription does not indicate that g is true. Circumscription shows us that g is not necessarily true,

even given that any default is true unless proved otherwise. Our version of Theorist, on the other hand, shows that the truth of g can be explained using different theories depending on the truth value of *red*.

The main complaint we have about Circumscription is the fact that the user often must provide a guess as to what the answer will be ("apply wishful thinking", as McCarthy says ([M86, p. 103])). In his example in [M86, section 10 and appendix A], McCarthy uses circumscription to consider what sorts of beings fly. After spelling out the facts of the world, the user *tells* the system the following:

$$\begin{aligned} flies'\, x &\equiv bird\, x \wedge \neg ostrich\, x \\ ab'\, z &\equiv [\exists x.bird\, x \wedge z = aspect1\, x] \\ &\quad \vee [\exists x.ostrich\, x \wedge z = aspect2\, x] \end{aligned}$$

The first in effect gives the answer, the second tells the circumscriber that it may assume that birds and ostriches *do* exist. McCarthy does not discuss what happens if the user guesses wrong; he does say that some circumscriptions will conclude that "there are no ostriches if none are explicitly mentioned." ([M86, p. 108])

Example 5 Compare the above with Figure 4. The facts given in the figure are the same as those given by McCarthy in his example. What is added is the default that things may be presumed not to be abnormal (which corresponds to minimising with respect to ab).

Note that in our extension of Theorist there is no need to "apply wishful thinking" nor to tell the system that the things we are discussing are actually to be considered to exist. Even when no birds or ostriches are known the system indicates appropriate conditions.

6 Future Work

This project is reasonably complete. The code does all that was expected of it. Perhaps improvements made to the Theorist code since [PGA86] was produced could be made to the code here.

One improvement that might be considered is to keep the conditions with the defaults that generated them. Consider the following example:

```
: m .abnormal;

fact    n(flies(X)) <- n(ab(notflying X));
fact    ab(notflying X) <- bird(X);
fact    flies(X) <- bird(X) n(ab(birdsfly X));
fact    ab(birdsfly X) <- ostrich(X);
fact    n(flies(X)) <- ostrich(X) n(ab(ostrichesdontfly X));

default n(ab(N X));

fixed   ostrich(X);
fixed   bird(X);
end;

: consult .abnormal;
: explain flies(tweety);
answer : flies(tweety)
theory : if     n(ostrich(tweety)) bird(tweety)
       then     n(ab(birdsfly,tweety))
no (more) answers

: explain n(flies(hannible));
answer : n(flies(hannible))
theory : if     n(bird(hannible))
       then     n(ab(notflying,hannible))
answer : n(flies(hannible))
theory : if     ostrich(hannible)
       then     n(ab(ostrichesdontfly,hannible))
no (more) answers
```

Figure 4: Outdoing the Circumscription

Example 6

$$\begin{aligned} F &= \{red \wedge ontable \wedge cleartop \Rightarrow g \\ &\quad heavy \Rightarrow \neg ontable \\ &\quad big \Rightarrow \neg cleartop\} \\ \Delta &= \{ontable,\ cleartop\} \\ \Phi &= \{red,\ heavy,\ big\} \end{aligned}$$

A response of the form

```
answer : g
condns : red n(heavy) n(big)
theory : ontable if n(heavy)
         cleartop if n(big)
```

might be more useful than our

```
answer : g
theory : if red n(heavy) n(big)
         then ontable cleartop
```

The changes required to implement this are mostly cosmetic, but will involve some extra work to keep track of conditions generated by more than one default.

What remains is to apply this system to various problems and see how useful it is. Poole ([P87]) suggests three uses for the system.

- in drawing up plans that depend on some condition which is not known now, but which may be determined at execution time.
- in making decisions where we want our decision to be appropriate regardless of factors we cannot determine the truth of.
- in making diagnoses, representing tests by fixed predicates and carrying out these tests only when the corresponding condition appears in the explanation.

Example 7 Figure 7 shows an example of using the system to do planning. The problem is to get home to Dartmouth from southend Halifax. Three

```
: m getting.home;

fact dartmouth <- barrington brunswick macdonald;
fact dartmouth <- barrington mackay rotary;
fact dartmouth <- robie mackay rotary;
fact n(rotary) <- hurry;
fact n(brunswick) <- traffic hurry;

default barrington;
default brunswick;
default macdonald;
default mackay;
default robie;
default rotary;

fixed traffic;
fixed hurry;
end;

: consult getting.home;
: explain dartmouth;
answer : dartmouth
theory : if     or([n(hurry),n(traffic)])
         then   macdonald brunswick barrington
answer : dartmouth
theory : if     n(hurry)
         then   rotary mackay barrington
answer : dartmouth
theory : if     n(hurry)
         then   rotary mackay robie
no (more) answers

: fact hurry;
: explain dartmouth;
answer : dartmouth
theory : if     n(traffic)
         then   macdonald brunswick barrington
no (more) answers
```

Figure 5: Going Home

routes are considered. Brunswick street is a very slow route to take if there is any traffic on it, while the rotary is slow most of the day. What we don't know yet is whether we will be in a hurry to get home and whether there will be a lot of traffic on Brunswick street.

We ask the system to plan conditional routes for us, which it does. Later we decide that we are in a hurry to get home. We consider the routes available to us. The latter two are rejected out of hand while the first is acceptable only if there is no traffic on Brunswick. If it turns out that there is traffic then we will be unable to achieve our goal of arriving home in a hurry. (Note that the second call to *explain* would have been unnecessary had we saved the three plans.)

7 Conclusions

This paper has described an extension to Theorist. The addition of fixed predicates allows for explanations of things that were previously unexplainable. The conditions of a theory under the new system may be viewed in two ways. One is that these conditions must hold before the theory applies. The other is to look upon them as an explicit statement of the side effects of our theory. Rather than being things that must be true before the theory applies, they can be viewed as things that will be true under our theory. They do not form an exhaustive list of such side effects since varying (non-fixed) predicates will not appear.

This system appears to do what Circumscription does without needing as much guidance. I feel that this makes it a better system than Circumscription for doing common-sense reasoning.

A The Code

A.1 explainer

The predicates in figure 6 are the code for conditional and unconditional explanation of a list of goals. Predicate *report* prints the theory to the screen. *btassert-all* asserts each element of its argument (a list) as facts, and retracts them again when backtracking occurs.

A.2 prover

Figures 7 and 8 show the *pr*, *prusedef*, *prusefix* and *all-pr* predicates, the main predicates in the prover module. Except for the addition of a new clause (and, of course, the extra arguments for conditions of the theory), *pr* is the same as it is in [PGA86]. The five- and four-argument versions of *pr* are similar to the six-argument version shown. The differences are:

1. Only previously used defaults are allowed to be used.
2. In the four argument version, only previously used conditions are allowed.
3. The user is not asked about predicates (*i.e.* the ask_user clause does not appear).

References

[L86] V. Lifschitz, "Pointwise Circumscription: Preliminary Report", *Proc. AAAI-86*, pp. 406–410.

[M86] J. McCarthy, "Applications of Circumscription to formalizing commonsense reasoning", *Artificial Intelligence 28*, 1986, pp 89–118.

[P86] D. Poole, *Default Reasoning and Diagnosis as Theory Formation*, Technical Report CS–86–08, Department of Computer Science, University of Waterloo, March 1986.

[P87] D. Poole, *Fixed Predicates in Default Reasoning*, Research Report CS–87–11, Department of Computer Science, University of Waterloo, February 1987.

[PGA86] D. Poole, R. Goebel, R. Aleluinas, *Theorist: a logical reasoning system for defaults and diagnosis*, Technical Report CS–86–06, Department of Computer Science, University of Waterloo, February 1986.

```
%       explain ( G D C ) means that G is conditionally explainable
%                         with conditions C on theory D.

explain ( G D C ) <-
        prall ( G [] [] D [] C );

%       explain  ( G ) gets a conditional explanation for G and prints
%                      it on the screen.

explain ( G ) <-
        explain ( G D C )
        report ( G [C D] )
        fail;
explain ( G ) <-
        write ( "\nno (more) answers\n" );

%       uncond-explain ( G ) gets an unconditional explanation for G
%                       and prints it on the screen.

uncond-explain ( G ) <-
        uncle ( G L )
        report ( G L )
        fail;
uncond-explain ( G ) <-
        write ( "\nno (more) answers\n" );

%       uncle ( G L ) builds a list of conditional explanations (L)
%                     of G until one with empty conditions is found.

uncle ( G [D] ) <-                    % empty conditions found
        explain ( G D [] );
uncle ( G [C D|L] ) <-                % no empty conditional...
        not ( explain ( G D [] ) )    %       add negation of
        explain ( G D C )             %       conditions to facts,
        and-neg ( C NC )              %       try again.
        btassert-all ( NC )
        uncle ( G L );
```

Figure 6: Module Explainer

```
%
%       pr ( G A D1 D2 C1 C2 ) means G is provable with ancestors A,
%                                starting defaults D1 and conditions C1,
%                                and final defaults D2 and conditions C2.

pr ( G A D D C C ) <-                    % search up tree for negation
        neg ( G GN )                     % of current goal
        member ( GN A )
        ;
pr ( G A D1 D2 C1 C2 ) <-                % retrieve next fact for
        fact ( G Body )                  % expanding proof tree
        prall ( Body [G|A] D1 D2 C1 C2 )
        ;
pr ( G A D1 D2 C1 C2 ) <-                % get next default/hypothesis
        default ( N G B )                % for expanding proof
        prusedef ( G A N B D1 D2 C1 C2 )
        ;
pr ( G A D1 D2 C1 C2 ) <-                % use a fixed predicate.
        fixed ( G )
        prusefix ( G A D1 D2 C1 C2 )
        ;
pr ( G A D D C C ) <-                    % G is defined in Prolog
        prolog ( G )
        prove ( G )
        ;
pr ( G A D D C C ) <-                    % ask the user
        askable ( G )
        ask_user ( G )
        ;

prall ( [] A D D C C )
        ;
prall ( [G|B] A D1 D3 C1 C3 ) <-
        pr ( G A D1 D2 C1 C2 )
        prall ( B A D2 D3 C2 C3 )
        ;
```

Figure 7: Module Prover

```
%       prusedef ( G A N B D1 D2 C1 C2 ) means that G is proved with
%                                        ancestors A, using default
%       name N, body B, starting defaults D1 and conditions C1, and
%       final defaults D2 and conditions C2.

prusedef ( G A N B D1 D2 C1 C2 ) <-      % N has already been
        member ( N D1 )                  % used in our proof
        prall ( B [G|A] D1 D2 C1 C2 );
prusedef ( G A N B D1 D2 C1 C3 ) <-      % N has not been used so must
        not ( member(N D1) )             % be proved consistent
        prall ( B [G|A] [N|D1] D2 C1 C2 )
        neg ( G GN )
        all-pr ( GN [] D1 C2 C4 )
        not ( member ( and([]) C4 ) )
        or-neg ( C4 C5 )
        add-fix-each ( C5 C2 C3 );

%       prusefix ( G A D1 D2 C1 C2 ) means G is proved with ancestors
%                                    A, starting defaults D1 and
%       conditions C1, and final defaults D2 and conditions C2.

prusefix ( G A D D C C ) <-              % previously used default
        member ( G C );
prusefix ( G A D D C1 C2 ) <-            % not previously used
        not ( member ( G C1 ) )          % must not be provably true
        not ( pr ( G [] [] [] ) )        % nor provably false
        neg ( G NG )
        not ( pr ( NG [] D C1 ) )
        add-fix ( G C1 C2 );

%       all-pr ( G A D C1 C2 ) means that C2 is the list of all
%                              conditions for G that do not follow
%       from C1 and the facts, given theory D and ancestors A.

all-pr ( G A D C1 C2 ) <-
        btassert-all ( C1 )
        assert ( doing ( G A D C1 [] ) )
        pr ( G A D [] C3 )
        retract ( doing ( G A D C1 L ) )
        assert ( doing ( G A D C1 [and(C3)|L] ) )
        fail;
all-pr ( G A D C1 C2 ) <-
        retract ( doing ( G A D C1 C2 ) );
```

Figure 8: Module Prover

Implementing Defaults with Connectionism

Brenda Parsons
Logic Programming and Artificial Intelligence Group,
University of Waterloo,
Waterloo, Ontario, Canada, N2L 3G1
bdparsons@waterloo.csnet

April 22, 1987

Abstract

The combination of two areas in Artificial Intelligence research, default reasoning and connectionism, leads to some particularly exciting solutions. We examine some of the classic multiple extension problems that have arisen in default reasoning and apply them to our connectionist model, the R-Network. The results are particularly startling, as the R-Network produces the desired results for this particularly small subset of problems which have plagued AI researchers for the past several years.

1 Introduction

We propose a network model which combines the connectionist model[Feld82] and the framework of the default reasoning system Theorist[Pool86b], which will solve a class of problems which lead to multiple extensions. We show that the results will simulate choosing the most specific theory in default reasoning.

One of the major problems with connectionist models to date has been determining weights to assign to connections between the nodes. In our model, the Reasoning-Network, or R-Network, we will assign the weights using the framework of Theorist, which will reflect our beliefs in the strengths of the hypotheses.

We shall start by describing the basic connectionist model, then we will describe the implementation of the R-Network. We assume that the reader is familiar with the multiple extension problem, as described by Reiter[Reit81],

Touretzky[Tour84a] and McCarthy[McCa86], as the details are beyond the scope of this paper. We shall show the R-Network solving some of the common default reasoning problems that give rise to multiple extensions. Lastly, we conclude that with further research, the R-Network will be able to simulate commonsense reasoning.

2 Description of the Connectionist Model

Connectionism, as described by Feldman and Ballard [1982], is based on storing the knowledge of the system in the connections between the computing units or nodes, much the same as the neural network of the brain.

How do humans carry out complex activities such as walking and talking at the same time? The nature of the brain's neural network is such that it can be thought of as a massively parallel and highly connected system of neurons.

2.1 Neural Structures versus Connectionism

The actual neural circuitry of the cortex is remarkably consistent. The individual neurons of the network are relatively unimportant since information is duplicated in many different neurons. This information is duplicated because the brain-cells die off.

The connectionist model has nodes or computing units which correspond directly to the individual neurons, but currently, the duplicity of the nodes is not used, as this would make the network very complicated to work with.

The neurons of the brain are connected by axons, which carry inhibitory and excitatory stimulus between the cells. This concept corresponds directly to the connections between the units in the connectionist model.

Neurons produce action potentials, brief spikes that propagate down axons, when the action potential reaches a synapse, the signal it produces in the post synaptic neuron rises to a maximum and then exponentially decays with the time constant on the membrane (typically around 5 milliseconds for neurons in cerebral cortex).

In a connectionist model, various methods of passing information between the nodes are used, but they are all based on sending excitatory or inhibitory impulses to all units that they are connected to. Each node in the model has an activation threshold which must be reached before it can fire or pass along information to its neighbours.

The 'cycle time' of a neuron is about two milliseconds or about 10^6 times slower than a computer. However, in a connectionist model, with all the nodes being synchronously updated, the cycle time will be close to that of a neuron.

The brain's connections are fixed, what may change are the weights on the connections. The same is true of the connectionist model. All the connections between the nodes are fixed upon the creation of the model. The tuning of the model is done by modifying the weights or strengths of the connections between the nodes. The fixed connections in the brain are what accounts for long-term learning and short-term associations can be handled by systems of dynamic bindings as described by Feldman[Feld82a]. The same is true in the connectionist model.

There is no symbol passing capability between the neurons in the brain, and no central controller telling the neurons what to do. The brain has on the order of 10^{11} processors, with 10^3-10^4 connections each. In the connectionist model, there is no controller watching over the units, so each runs as its own process. Any information that is passed between the units is in the form of excitatory or inhibitory pulses.

All encodings in the brain are in terms of the relative strengths of synaptic connections. One premise is that individual neurons do not transmit large amounts of symbolic information, rather, they compute by being 'appropriately connected' to large numbers of similar units[Feld82]. This, in essence, is the basis of the connectionist model. All the information is stored in the weights or strengths of the connections between the units.

2.2 The Structure of the Unit

The basic computational *unit* of Feldman & Ballard's connectionist model has the following properties [Feld82]:

- {q} – a set of *discrete states*, < 10
- p – a continuous value in the range [-10,10], called *potential* with the accuracy of several digits
- v – an *output value*, integers $0 \leq v \leq 9$
- i – a vector of *input values*, $i_1, ...i_n$

The restriction that 'v' take on small integers is central to the concept of the model. The firing frequencies of neurons range from a less than ten to a

few hundred impulses per second. In the one-tenth of a second needed there can only be a limited amount of information encoded in frequencies; so the ten output values attempt to capture this concept.

The state set q has biological references to accommodate models of fatigue, peptide modulations and other qualitative state changes. On the computational side, the discrete states permit the use of analysis and proof techniques from computer science.

The potential is a crude form of memory – an abstraction of the instantaneous membrane potential that characterises neurons.

The standard model of the individual unit may be used to model anything from a small part of a neuron to the external functionality of a major subsystem. Since there is no central controller for the connectionist model, each unit is similar to a stylised neuron which broadcasts its outputs to all connected units or neighbours.

To broadcast to all of its neighbours, the basic computational ***unit*** uses functions which transform the old values to new values as follows:

- $p \leftarrow f(i, p, q)$ new potential
- $q \leftarrow g(i, p, q)$ new state
- $v \leftarrow h(i, p, q)$ new output

The form of the functions f, g, and h may vary, but in general they will be restricted to conditionals or simple functions that will compute continuously. The function which modifies the input determines how inputs to a unit are to be summarised. The potential update function f, defines the relationship between the inputs and the unit's overall potential.

The function h, that updates the output determines the relationship between the unit's potential and the actual value the unit transmits to its neighbours. The value output by a unit is used as a measure of the belief in the truth of the hypothesis which is represented by the unit.

The state update function g, depends on the values of the the unit's potential and the output value that unit transmits to its neighbours.

So, in summary, the technique used to encode information in a connectionist network is called the unit/value principle where a unit represents the value or hypothesis of a parameter[Feld82]. A unit receives input from other units which provide positive or negative evidence for the hypothesis the unit represents. If this input exceeds the unit's threshold, the unit will fire at an output rate that expresses a confidence measure which is, in turn, transmitted to all connected units.

3 The Reasoning Network

Starting from the basic connectionist model described in section 2, we expand this model by defining functions which govern the values of the potential and the current state of the node.

In the R-Network, we generally compute the potential value of a node by summing all the values of the inputs from adjoining nodes. More formally, $P_t = P_{t-1} + \sum W_i$ where P_t is the new potential at time t, and W_i is the modified weight on the connection from node i. All nodes in our model have an output of one.

We have defined four states S, that any node may be in, namely, 'dormant', 'active', 'decay', or 'stable'.

All nodes will start with a potential of zero, and in an initial state of 'dormant'. A node can become 'active' in two ways. One method is to externally (or manually) set the potential of the node. This is how we start a simulation. Alternatively, within a simulation, a 'dormant' node's potential, updated as described previously, reaches a preset activation threshold α, causing the node to become 'active'.

If the node has become 'active', one of two things can happen. If the potential is increasing from zero, eventually, it will reach a stability threshold β, and the node will become 'stable'.

Once a node becomes 'stable', it remains 'stable' as does its potential for successive iterations, ie. $P_t(X) = P_{t-1}(X)$. The other possibility is that an 'active' node will have a potential that does not change over k iterations. If this occurs, the node is said to go into a state of 'decay'.

A node which is decaying will use a modified updating function for calculating its new potential, namely, $P_t(X) = \|P_{t-1}(X)\| - \lambda$, where λ is the decay rate. Upon reaching the activation threshold α, the node X will once again become 'dormant'. These updating criteria are presented formally in Definition 1.

Definition 1 $\forall$ nodes X at time t

- *if* $S_t(X) \equiv$ '*dormant*' *and* $P_t(X) > \alpha(X)$ *then*
  $S_t(X) \equiv$ '*active*'
- *if* $S_t(X) \equiv$ '*active*' *and* $\|P_t(X)\| > \beta(X)$ *then*
  $S_t(X) \equiv$ '*stable*'
- *if* $S_t(X) \equiv$ '*active*' *and* $P_t(X) = P_{t-1}(X)$ *for* k *iterations then*
  $S_t(X) \equiv$ '*decay*'

- *if* $S_t(X) \equiv$ *'decay' and* $P_t(X) < \alpha(X)$ *then*
  $S_t(X) \equiv$ *'dormant'*

4 Theorist and the R-Network

We now describe how to map the Theorist representation of a problem onto the R-Network using the standard 'birdsfly' problem described in Example 1.

Example 1
Let

$$\Delta = \{\ flies(X) \leftarrow bird(X)$$
$$\neg flies(X) \leftarrow emu(X)\}$$

and let

$$F = \{\ bird(X) \leftarrow emu(X)$$
$$feathers(X) \leftarrow bird(X)$$
$$emu(tweety)\}$$

To build the R-Network, we define the correlation between the representations so that we can create the nodes, the connections, and assign appropriate weights reflecting our beliefs in the strengths of the connections.

4.1 Creating the nodes

Our first task is to generate the nodes. We create nodes for each predicate and constant symbol, with corresponding names, similar to Feldman and Ballards unit: flies, bird, emu, feathers and tweety.

4.2 Creating the Connections

The translation rules used to generate directed connections for the R-Network are as follows:

1. ground literals, such as $emu(tweety)$ are connected from the constant to the predicate symbol (ie. tweety$\rightsquigarrow$emu).

2. rules or wffs, such as $bird(X) \leftarrow emu(X)$ are connected from the right to the left (ie. emu(X)$\rightsquigarrow$bird(X)).

3. constants connect to themselves (ie. tweety$\rightsquigarrow$tweety).

4. facts that are direct inheritances, such as $feathers(X) \leftarrow bird(X)$ and $bird(X) \leftarrow emu(X)$, connect from the bottom to the top of the inheritance. (ie. emu$\rightsquigarrow$feathers)

Rules one and two are straightforward. Rule three is used to create a positive feedback so that the constants do not decay out of the network[Dell85]. Rule four is used to limit the overhead of the model. There are also implicit non-directed connections between all the nodes in the model. From example 1, the connections created are shown in Table 1, using the indicated rules.

#	Connection	Rule	Initial Weight	Modified Weight
1	emu$\rightsquigarrow$feathers	4	F	F × 3
2	emu$\rightsquigarrow$flies	2	$\neg\Delta$	$\neg\Delta\times$ 3
3	emu$\rightsquigarrow$bird	2	F	F × 3
4	tweety$\rightsquigarrow$emu	1	F	F × 2
5	tweety$\rightsquigarrow$tweety	3	F	F × 2
6	bird$\rightsquigarrow$flies	2	Δ	$\Delta\times$ 2
7	bird$\rightsquigarrow$feathers	2	F	F × 2

Table 1: R-Network Connections for the 'birdsfly' example

4.3 Assigning the Initial Weights

The objective of our network is to reflect the strengths of our beliefs in the relationships between the nodes in the model. Hence, we will assign the weights according to the framework of the Theorist system.

In Theorist, we would like to believe facts with a stronger conviction then we want to believe for defaults. The same is true for the R-Network, the weights on the connections are greater (distanced from zero) for facts then defaults. When representing a fact in our model, we assign it a higher numerical value. If the fact displays the negation, as does the statement $\neg flies(X) \leftarrow emu(X)$, then we reflect this with a negative numerical value.

We also want to be able to let the system 'learn'. We've added the symbol M to describe the 'possible' knowledge gained from a learning experience and the symbol φ to describe the absence of knowledge. Table 2 describes the belief system of the R-Network in relation to Theorist as well as the with pseudo-values that will be used in the calculations of the network.

Beliefs	Theorist	Symbols	R-Network Value
Total Belief	Fact	F	4
Believe True	Default	Δ	2
Possibly True	N/A	M	1
No Belief	N/A	φ	0
Possibly False	N/A	$\neg$M	-1
Believe False	$\neg$Default	$\neg\Delta$	-2
Total Disbelief	$\neg$Fact	$\neg$F	-4

Table 2: R-Network Belief System

From this table, it is clear that knowledge which is factual has a stronger representation in our mode then knowledge which we hold as a default. Additionally, the knowledge that we believe possibly true or false is weighted even lower than default knowledge.

In our example, connection number six from Table 1 will receive an initial weight of a default from Table 2, and connection number two will receive the weight of a negated default. The rest of the connections will receive positive fact weightings.

4.4 Modifying the Weights

To represent inferential distances[Tour84a] we modify the weights using a simple algorithm. We multiply the weighting factors on each outgoing connection, by the total number of outgoing connections from that node. For example, emu has three outgoing connections, to flies, bird and feathers, thus the weights on those connections will be multiplied by a factor of three. The complete R-Network for Example 1 is shown in Figure 1 with the nodes, connections and modified weights labelled accordingly.

4.5 The Simulation

Once a problem has been transformed to the R-Network model, activation of the network must take place. This is done manually by sending a excitatory pulse to each of the nodes in question. In Example 1, the goal is $?flies(tweety)$, or, we are asking for information about the connection tweety$\rightsquigarrow$flies. To start, we manually set the potential for both nodes to a preset value (in our case 10).

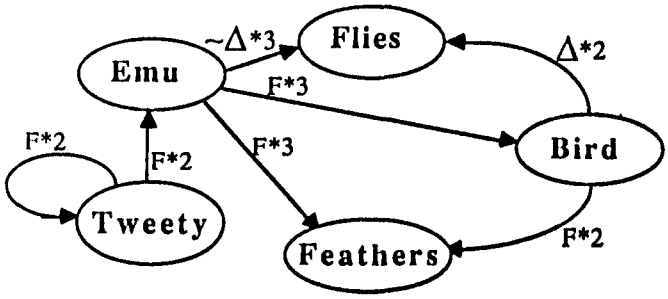


Figure 1: R-Network for flies(tweety)

In Table 3, each node in the network is represented by a tuple at each cycle. The first element of the tuple is the current state of the node. The second is the current potential and the third is the value of $\sum W_i$. We can see that in cycle 0, the nodes flies and tweety, are in an 'active' state, and both have the preset potential of 10. By applying the updating functions described previously, we see the results in cycle 1. In which emu has also become 'active', with a potential of 8. By the end of cycle 2, tweety has stabilised, and the rest of the nodes are 'active'. After cycle 3, all the nodes except flies have stabilised positively. It will take until cycle 15 for flies to stabilise negatively.

Cycle	Tweety	Emu	Bird	Flies	Feathers
0	(A,10,0)	(D,0,0)	(D,0,0)	(A,10,0)	(D,0,0)
1	(A,18,8)	(A,8,8)	(D,0,0)	(A,10,0)	(D,0,0)
2	(SY,26,8)	(A,16,8)	(A,12,12)	(A,4,-6)	(A,12,12)
3	(SY,26,8)	(SY,24,8)	(SY,24,12)	(A,2,-2)	(SY,32,20)
⋮	⋮	⋮	⋮	⋮	⋮
15	Yes	Yes	Yes	No	Yes

Table 3: Simulation of 'Birds Fly'

4.6 Analysis

After the system has reached a stable coalition, in our case fifteen cycles, the results can be analysed. In our example, our original query was $?flies(tweety)$. The result being that tweety has a positive value, and flies a

negative one. This can be seen as equivalent to the assertion $\neg flies(tweety)$.

Now, unlike Theorist, we can modify our R-Network to incorporate this new information. We want to modify the weight on the connection corresponding to $\neg flies(tweety)$, ie.tweety$\rightsquigarrow$flies, to have the initial weight of $-M$, indicating that we believe that the relationship reflects a possibly false situation. We must also modify the multiplier on the other outgoing connections from tweety to be three. The modified network is shown in Figure 2.

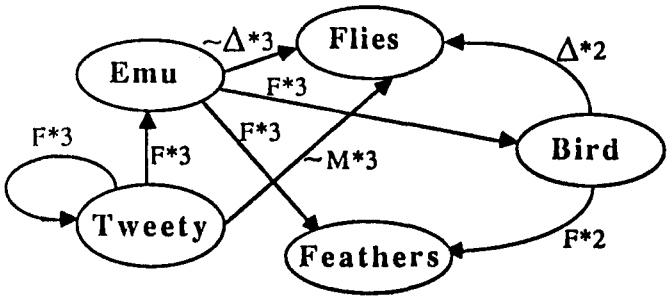


Figure 2: R-Network for modified flies(tweety)

4.7 Relevant Side Effects

In our simulation, we also inadvertently gained information about the previously un-weighted connections tweety$\rightsquigarrow$feathers and tweety$\rightsquigarrow$bird. Both have positive correlations and we could add the connections in the same manner as described previously. This information may not be relevant at the current time, but we store it away for future reference, which is analogous to the real learning process. Since the connections between nodes are already implicitly in existence, we can update the network, as described previously.

When we added the connection tweety$\rightsquigarrow$flies, the system stablised in six cycles, as compared to the fifteen cycles required previously. When we added the other connections, the system only took five cycles to stabilise. Thus, it appears that our model has learned from previous experiences.

5 Examples

We will now present a few more of the common default reasoning problems with to multiple extensions. In each case, we shall show how the R-Network resolves the multiple extensions. All examples given in the paper have been implemented using the University of Rochester's Connectionist Simulator [1][Fant86].

5.1 Cephalopods

The following example, has been used in current research to establish the correspondence between inheritance hierarchies with exceptions and default logic.[Ethe83] Etherington and Reiter used a network representation with five link types under the framework of Fahlman's NETL[Fahl83].

The example is given in terms of the Theorist model as follows:

Example 2
Let

$$\Delta = \{\ shell\_bearer(X) \leftarrow mollusk(X),\ \neg shell\_bearer(X) \leftarrow cephalopod(X)\}$$

and let

$$F = \{\ mollusk(X) \leftarrow cephalopod(X),\ cephalopod(X) \leftarrow nautili(X),\ cephalopod(honey),\ shell\_bearer(X) \leftarrow nautili(X)\}$$

For our purposes, we translate the Theorist representation into an R-Network, by creating the nodes shell_bearer, mollusk, cephalopod, nautili and honey. Then, using the methods described previously, we create the connections between the nodes, and assign the weights to the connections as shown in Figure 3.

5.1.1 Analysis

In this example, the goal is $?shell\_bearer(honey)$. As in the previous example, we manually set the potential of shell_bearer and honey, and start the simulation.

[1]Available for general use under /u/bdparsons/connect on watdragon

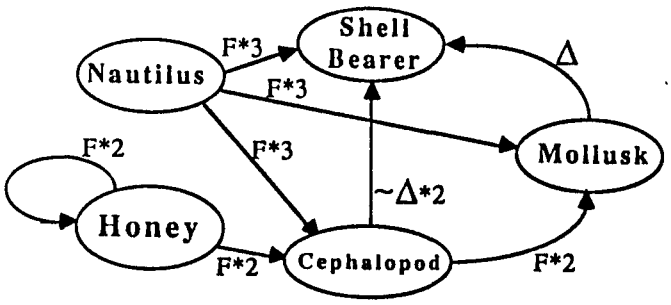


Figure 3: R-Network for shell_bearer(honey)

We are omitting the actual simulation table in effort to point the reader towards the intuitive information that can be gained from examining the pictorial representation of the problem.

We can explain the goal *shell_bearer(honey)* by following the arrows through the model giving us T_1 = {honey $\rightsquigarrow$ cephalopod $\rightsquigarrow$ mollusk $\rightsquigarrow$ shell_bearer}. But, we can also explain $\neg$*shell_bearer(honey)* by following T_2 = {honey $\rightsquigarrow$ cephalopod $\rightsquigarrow$ shell_bearer}.

Here we have two extensions. Which path do we choose? Which do we prefer? From Table 3, we can see that the incoming connections to the node shell_bearer are in direct competition. However, because of the weighting on the connection from cephalopod, the potential of shell_bearer will steadily decrease to a negative stable state. Thus, the R-Network will prefer T_2 as a solution.

We can add the new connection represented by $\neg$*shell_bearer(honey)* with the weighted value of $\neg$M as done previously. If we try to explain the goal *shell_bearer(honey)*, we will have the direct path T_3 = {honey $\rightsquigarrow$ shell_bearer}. However, T_1 and T_2 will still be activated, but the competition on shell_bearer will have a greater negative influence, only to produce the same result quicker. Once again demonstrating the learning mechanism of our model.

In runs of the simulator, the first run stabilises in sixteen cycles, and when modified, only seven cycles were required.

5.2 Bats Fly too

All the examples presented until this point have preferred the negative alternative of the multiple extensions. This example, shows that positive answers are possible. As always, the Theorist representation of the problem follows:

Example 3
Let

$$\Delta = \{\ flies(X) \leftarrow bat(X),\ \neg flies(X) \leftarrow mammal(X)\}$$

and let

$$F = \{\ mammal(X) \leftarrow bat(X),\ bat(bert)\}$$

Translating this example into terms for the R-Network we create the nodes, connections and weights as shown in Figure 4.

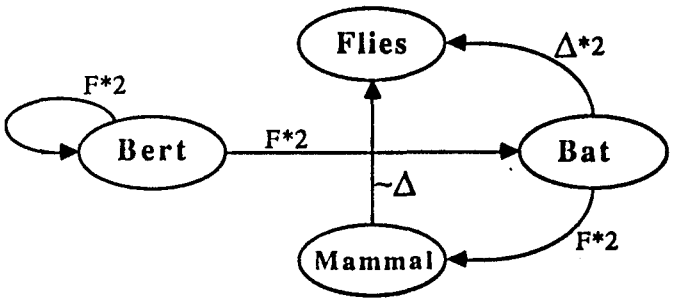


Figure 4: R-Network for flies(bert)

As can be seen, the conflicting theories will arise when dealing with the node flies. The dominating hypothesis or incoming connection is from bat with a weighting of $\Delta \times 2$, which out performs the weighting from mammal.

Since all the nodes in the network stabilised positively, we can add the link representing *flies(bert)* to our model. The original simulation stabilised in six cycles. When the connection was modified, it only took a mere four cycles to stabilise.

5.3 Is he or isn't he?

Example 4
Let

$$\Delta = \{\ pacifist(X) \leftarrow quaker(X),\\ \neg pacifist(X) \leftarrow pro\_defense(X),\\ pro\_defense(X) \leftarrow republican(X)\}$$

and let

$$F = \{\ quaker(nixon),\\ republican(nixon)\}$$

Reiter and Criscuolo (R&C), use the *Nixon-is-a-Pacifist* example to describe interacting defaults[Reit81]. Touretzky used the same example to prove his point about inferential distances[Tour84a]. R&C say that "intuitively we want to make no assumptions about his warlike nature" and NETL incorrectly produces the answer that nixon is a pacifist using the shortest path heuristic. Theorist deduces that it cannot determine the warlike nature of Nixon.

The Theorist representation was shown in Example 4 and the R-Network representation is shown in Figure 5.

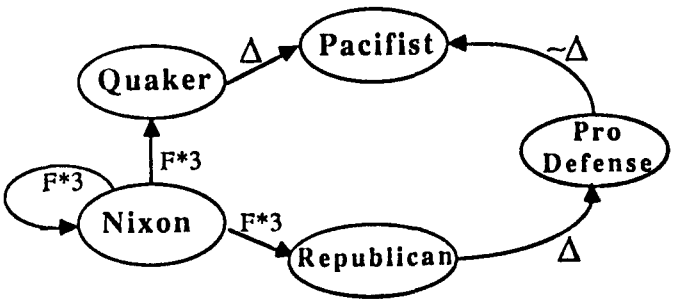


Figure 5: R-Network for pacifist(nixon)

Upon presenting the R-Network with the query ?*pacifist*(*nixon*), we find that there are two paths: T_1 = {nixon $\rightsquigarrow$ quaker $\rightsquigarrow$ pacifist} and T_2 = {nixon $\rightsquigarrow$ republican $\rightsquigarrow$ pro_defense $\rightsquigarrow$ pacifist}. However, in this case, the weightings on the incoming connections to pacifist are equal, and hence, the R-Network will eventually cause pacifist to decay and reach a 'dormant'

state. Since pacifist was initially activated and has stabilised as 'dormant', we deduce that there is not sufficient evidence to make a positive or negative decision.

The actual simulation of this example took twelve cycles to stabilise, most of which were used to decay pacifist into a 'dormant' state.

5.4 Tweety's Alive

This example demonstrates the R-Network's ability to handle conjunctions. In Theorist terms the problem is as follows:

Example 5
Let

$$\Delta = \{\ flies(X) \leftarrow bird(X) \wedge alive(X),\ \neg flies(X) \leftarrow emu(X)\}$$

and let

$$F = \{\ bird(X) \leftarrow emu(X),\ emu(tweety),\ alive(tweety)\}$$

The nodes are generated as before with one exception. If a conjunction exists in the Theorist representation, we create an extra node representing the conjunction. The nodes created for this example would be: bird, alive, flies, emu, tweety and finally alive_and_bird, as shown in Figure 6.

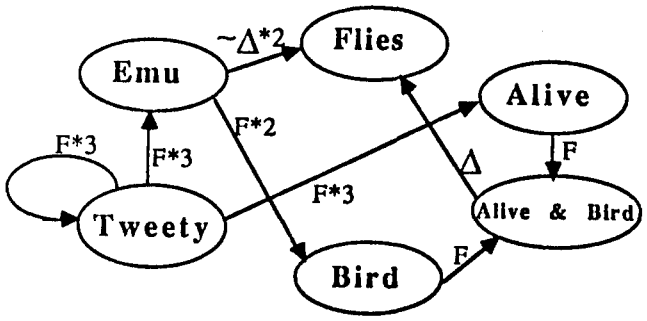


Figure 6: R-Network for conjunction example

The connections are generated as usual, taking note that the connection is alive_and_bird⇝flies, and not alive⇝flies together with bird⇝flies as might be expected.

The competing incoming connections to the node flies come from emu and alive_and_bird, in which emu has a greater weighting. It should be noted that alive_and_bird does not become 'active' until both alive and bird are active. Thus, when flies stabilises negatively it will lead to the preferred hypothesis $\neg flies(tweety)$. The actual simulation took 12 cycles to stabilise, and once the newly acquired information was added, the network took 6 cycles to stabilise.

6 Future Work

The R-Network uses relatively simple updating algorithms for maintenance of the network. With proper evaluation, the complexity of these functions could be increased, yielding a more stable and reliable model.

One step towards this goal would be to consider how to incorporate variables into the network so we can generate the actual answers to queries rather than just yes or no answers.

7 Conclusions

We have examined some of the problems that give rise to multiple extensions in default reasoning. It has been adequately shown that for this particular class of examples, the R-Network, provides a viable solution by preferring the most specific theory.

The R-Network seems to be oblivious to the current world, and relies wholly upon the beliefs of the underlying structure of the network. This corresponds directly to the neuronal methods of deduction. The R-Network places all its strength on the connections in the network. If the connections are incorrect, or the weights on the connections are wrong, as in our belief is incorrect, then the wrong answer will result. But, the same is true for any default reasoning system, including humans.

Since the R-Network has the capability of running in a massively parallel system, and the relatively few time steps or cycles that are required to find a solution, perhaps we have found a suitable model for commonsense reasoning.

8 Acknowledgements

Special thanks go out to all of you who put up with and answered all my stupid questions (you know who you are). To David for supplying the opportunity, and my boss Peter who is still mystified at how little work I get done for all the time I spend on the computer. And, Eric 'Bad Dog' Neufeld, for his valuable comments on earlier drafts of this paper. This work was not funded by any official grants, but managed to scrounge off the goodness of a lot of people's hearts.

References

[Cott85] G.W. Cottrell, *A Connectionist Approach to Word Sense Disambiguation*, Ph.D. Dissertation, Department of Computer Science, University of Rochester, May 1985.

[Dell85] G.S. Dell, "Positive feedback in hierarchical connectionist models: Applications to language production", *Cognitive Science*, Vol 9, 1985, pp. 3–23

[Ethe83] D.W. Etherington and R. Reiter, "On Inheritance Hierarchies With Exceptions", In *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., August 1983, pp. 104–108

[Fahl83] S.E. Fahlman, G.E. Hinton, and T.J. Sejnowski, "Massively parallel architectures for AI: NETL, Thistle and Boltzmann machines", In *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., August 1983, pp. 109–113

[Fant86] M. Fanty and N. Goddard, *User's Manual Rochester Connectionist Simulator*, Draft, Department of Computer Science, University of Rochester, Rochester, New York. 1986

[Feld82] J.A. Feldman and D.H. Ballard, "Connectionist models and their properties", *Cognitive Science*, Vol 6, 1982, pp. 205–259

[Feld82a] J.A. Feldman, "Dynamic connections in neural networks", *Biological Cybernetics*, Vol 46, 1982, pp. 27–39

[McCa86] J. McCarthy, "Applications of Circumscription to Formalizing Common-Sense Knowledge", *Artificial Intelligence*, Vol 28, 1986, pp. 89–116

[Pool86b] D.L. Poole, R.G. Goebel and R. Aleliunas, *Theorist: a logical reasoning system for defaults and diagnosis*, RR-CS-86-06, University of Waterloo, 1986

[Reit81] R. Reiter and G. Criscuolo, "On Interacting Defaults", In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC., Aug, 1981, pp. 270–276

[Tour84a] D.S Touretzky, "Implicit Ordering of Defaults in Inheritance Systems", In *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, August 1984, pp. 322–325

[Tour84b] D.S Touretzky, *The Mathematics of Inheritance Systems*, PhD Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa. May, 1984

Nested Default Reasoning

Term Paper for CS786
Prof. D. L. Poole
April 27, 1987

Paul Van Arragon

ABSTRACT

We discuss the use of a nested version of Theorist. It allows us to represent the reasoning of agents who reason about other agents. We first make several simplifying assumptions, and present a simple version of Nested Theorist. We demonstrate its use on the two-level "robot-in-a-pet-shop" scenario. A distinction is made between predicting and explaining. Finally, we discuss the difficulties that arise when we attempt to build a version of Nested Theorist where the assumptions have been relaxed. WARNING: This paper requires its reader to think about thinking about thinking.

Once upon a time, there was a king who was fond of playing mind-games with his subjects. So he brought out his five royal hats: three of them black and two white. He summoned three of his subjects, and put blindfolds on each of them. Then he placed a royal hat on each of their heads and removed their blindfolds. Since the subjects were standing in a row, all facing one direction, each subject could only see the royal hats of the subjects standing in front of him. That is, one subject could see both of the other two subjects; one subject could see only one other subject; and one subject couldn't see anyone. None of them could see their own royal hat. The king told the subjects that their royal hats were chosen from a collection of three black ones and two white ones, and then asked the subject who could see two royal hats whether he could figure out what colour royal hat he was wearing. "No," he replied. The king asked the subject who could see one royal hat. "No." Finally, the king asked the third subject, who couldn't see any royal hats, and his astute answer was, "Yes."

What colour was the third subject's royal hat?

A similar illustration was used by Konolige [1985] to demonstrate that sometimes in order to solve a problem, we must reason about other people's beliefs. The third subject could only deduce the colour of his royal hat by understanding why the other two subjects could not deduce the colour of theirs.

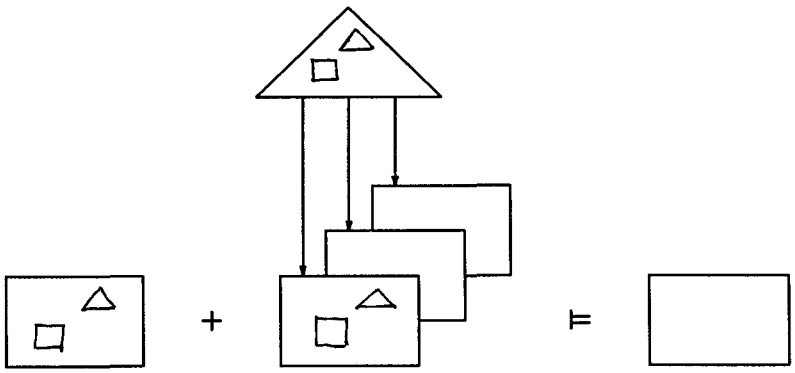
Things can get more complicated than the illustration conveys, however. Sometimes, not only must agents reason about someone's knowledge, but also they must reason about someone's *default* knowledge. Imagine that a robot rolls into a pet store. It has recently bought a pet emu, and has come to buy a cage, to prevent the emu from running all over the robot's house. The robot asks, "Can I buy a bird cage?" Now, the salesperson, not knowing that the bird to be caged is an emu, starts showing the robot the store's collection of closed bird cages. Actually, the robot would rather have a cage without a top on it. Since emus don't fly, he really only needs a closed fence. Hence, the robot reasons about the salesperson's beliefs, and adds, "My bird is an emu."

We have a system, Theorist [Poole, 1986], that models default reasoning, but how could we use it to model nested belief, as above? In section 1 we present a simple modification of Theorist to allow nested reasoning. We demonstrate its use on the pet-shop example, and discuss some problems that arise: namely, that we have to test

consistency on more than one level, and that predictions that we want to make with the system are not necessarily in any extension. Section 2 describes what we mean by using the system for explaining, as opposed to predicting. A semantics and an algorithm for explaining are presented. Section 3 briefly discusses a more complex problem that the version of Nested Theorist we present doesn't handle, and section 4 suggests further research.

1. The Simple Nested Framework

The underlying idea is to allow one agent to model another agent. Just as we can model an agent's default reasoning with a Theorist system, we can model an agent's reasoning about another agent as a Theorist system which reasons about a Theorist system. Hence, we require a nested Theorist system, where the inner system corresponds to an agent whose reasoning is being modeled, and the outer system corresponds to the agent doing the modeling. (See Figure 1, and cf. [Poole, 1986] Figure 1.)



A Nested Theorist

Figure 1

We attempt to define the semantics of this two-level system: informally, we can explain a goal g if we can find a consistent set of facts and defaults of the outer system, such that the Theorist system that these facts and defaults comprise (the inner system) explains g. The semantics of the inner system is identical to the semantics of a non-nested Theorist. We give an example before we write out the semantics more formally.

1.1. The Pet Store

Consider the predicament of the robot in the pet store. In his view of the salesperson, the salesperson believes that the pet bird can fly. According to the robot, the salesperson knows that this bird (let's call it Tweety) is a bird, and the salesperson assumes that Tweety flies, because birds typically fly.

The robot's view of the situation is different. It knows that Tweety is an emu. Hence, the robot's view contains the following:

- Tweety is a bird,
- Birds typically fly,
- Emus are birds,
- Emus typically do not fly, and
- Tweety is an emu.

We introduce some syntax to state all of the above more precisely and compactly.

Let $F_a p$ mean that agent a holds p as a fact. Agent a always holds p as true.

Let $\Delta_a p$ mean that agent a holds p as a default. Agent a holds p as true as long as p can be consistently believed.

Let r refer to the robot, and let s refer to the salesperson.

For example, "the robot believes as a fact that the salesperson believes by default that birds fly" is represented by $F_r\Delta_s(f(X)\leftarrow b(X))$. (The predicate f stands for "flies," and b stands for "is a bird.")

The above situation in the pet store is represented by the clauses in figure 2. Note that the only difference between the two views is that the view on the right is missing the fact *e(tweety)*. Other clauses concerning emus exist, but cannot be used to derive *f(tweety)*. Of course, the salesperson may not be thinking about emus at all, but it doesn't hurt to include propositions about emus in a representation of the salesperson's view, since if we ask the salesperson about emus, we expect him or her to know that they don't fly.

This example is straightforward. To see what the robot will predict about what the salesperson predicts with respect to flying, we determine what the relevant axioms predict. These axioms are the ones subscripted by r and s, in the right half of figure 2. They predict that Tweety flies, using the theory that birds fly.

Turning to the robot's prediction about flying, we see that the extra fact, $e(tweety)$, causes its model to predict $\neg f(tweety)$. (Note that we would have encountered the problem of multiple extensions here [McCarthy, 1986], if $\neg f(X)\leftarrow e(X)$ were a default instead

r's view	r's view of s's view
$\Delta_r(f(X) \leftarrow b(X))$	$\Delta_r\Delta_s(f(X) \leftarrow b(X))$
$F_r b(tweety)$	$F_r F_s b(tweety)$
$F_r(\neg f(X) \leftarrow e(X))$	$F_r F_s(\neg f(X) \leftarrow e(X))$
$F_r(b(X) \leftarrow e(X))$	$F_r F_s(b(X) \leftarrow e(X))$
$F_r e(tweety)$	

Clauses Representing the Pet Store Example

Figure 2

of a fact. Since this problem exists independently from this nested framework, we will not attempt to solve it, but instead we use axioms which have a single extension.)

1.2. Testing Consistency at Various Levels

A difference from the non-nested version of Theorist is that we now are dealing with defaults on more than one level. We are not just assuming that birds fly, but are also assuming that the salesperson is assuming that birds fly. In making one assumption, we are actually making two assumptions. Each of these must be tested for consistency, since they are used on either level if consistent.

We may find, however, that it is always consistent for an agent to make an assumption about another agent's defaults. How could it be inconsistent for one agent to assume another agent has the default that bird's fly? The only clause that is directly inconsistent is $\neg\Delta_{agent}(f(X) \leftarrow b(X))$, but such a clause is not allowed in our syntax. Specifying $\neg\Delta_i p$ makes no sense in our semantics, as it doesn't in the original Theorist. We can only state what hypotheses exist, not which don't exist. Perhaps we should be allowed to state such negative information, but then we would have to reconsider the original semantics. The ramifications are not clear.

Another way that the hypothesis of a default might be regarded as inconsistent is if the observations contradict it. Perhaps one has observed that an agent thinks Tweety is a bird, and that Tweety does not fly. Then isn't it inconsistent to attribute the agent with the default belief that birds fly? Unfortunately, not. As in the pet-shop example, the agent could have a fact such as "Tweety is an emu." If so, it is consistent to believe that the agent has the default that birds fly, as long as the default doesn't apply to Tweety. And, as we might expect, even if a theory of an agent predicts an observation incorrectly, we cannot say that a part of it (a particular default) is inconsistent.

Perhaps we could isolate the predictive power of a default, and thereby prove it inconsistent. If, for example, we introduce a new individual, X, and tell the agent that X is a bird, and then ask the the agent whether X flies, and the agent replies "no," we have isolated the default as much as possible. By introducing to the agent a bird, X, with no other connotations, we can find out whether the agent operates with the default that birds fly. The answer "no," means the default doesn't exist. It may be very difficult to do this in practice, however. It is hard to avoid connotations as we have attempted.

The lesson to be learned by this is one that we could have expected. When a theory is found to be inconsistent with respect to observations, we cannot single out a part of the theory to be at fault. Only a theory as a whole can be proven inadequate. A new theory must be proposed, but the new theory may contain defaults from the previous theory.

1.3. Predictions Outside of Extensions

Related to this difficulty of proving a default inconsistent, is another serious problem. We encounter it when we realize that a model of an agent sometimes must include a proper subset of the available defaults about the agent. For example, in the pet shop, the robot must be prepared to hypothesize that the salesperson knows that Tweety is an emu. After the robot utters "My bird is an emu," the robot's model of the salesperson changes to include $\Delta_r F_s e(tweety)$. But if we make available the hypothesis $\Delta_r F_s e(tweety)$, we must also find a way to reject it as we model the initial situation (where the salesperson doesn't know that Tweety is an emu). The initial situation is modeled using a proper subset of the available defaults. But, as we discussed above, we may have no way to prove the excluded hypothesis inconsistent. If it is not inconsistent, how can we reject it?

Here we notice a strange property that accompanies the attempt to build a nested Theorist: the goals that can be explained by a subset of the available defaults are not necessarily explainable by the full set of defaults. That is, adding more defaults will cause us to lose former theorems. In the original Theorist, adding *facts* caused theorems to disappear. This is known as nonmonotonicity. Nested Theorist is nonmonotonic with respect to adding *defaults* as well.

The pet shop example illustrates this. Adding the default that the salesperson knows that Tweety is an emu, $\Delta_r F_s e(tweety)$, no longer allows us to derive that the salesperson believes that Tweety flies. $F_s e(tweety)$ is consistent with our observations (since we haven't made any observations), and so we must add it to our model of the

salesperson. The default $\Delta_e(f(X) \leftarrow b(X))$ is now inconsistent, and therefore retracted. Hence, our model of the salesperson predicts that tweety doesn't fly.

Therefore the proof theory of Theorist (using just as many defaults as are necessary to derive something) no longer works. A theorem of the Theorist proof theory was that anything that can be explained with a subset of the defaults is still proven with all the defaults added. That is, we can only prove things that are in an extension [Poole, 1986]. Now we want to prove things that aren't in an extension. To derive that Tweety doesn't fly, we must use *all* the defaults. That is fine, but if we use all the defaults, how will we represent the case where the salesperson doesn't know Tweety is an emu?

2. Explaining

A solution to the above problem provides motivation for distinguishing *predicting* from *explaining*. The above discussion of the pet shop example concerned predicting. That is, we discussed how a model of an agent could predict what the agent would say when asked a question. The robot's model of the salesperson predicted that the bird could fly. The robot's model predicted that the bird couldn't fly.

To acquire these models in the first place, we must find a model that can *explain* all our observations. The same distinction arose with the non-nested Theorist. Explaining an observation by building a theory is different from predicting consequences of a theory already built. To build the theory in the first place, we must find a theory that predicts exactly our observations (i.e., a theory that is an *explanation* of these observations).

In the pet shop example, however, we didn't discuss the existence of any observations. Why then do we assume that the robot knows that Tweety is an emu, but that the salesperson doesn't? It seems that we need also to make a distinction between *prototypes* and *defaults*. Prototypes are things that we assume only if we have evidence for them; defaults are always assumed if consistent. The hypothesis that someone knows that Tweety is an emu is like a prototype. It must be that we had evidence to assume that the robot knew this. The evidence could be that the robot saw the emu when it was originally bought. We didn't have any evidence that the salesperson knew Tweety was an emu, however. That Tweety is a bird is also a prototype, but we have evidence that both the salesperson and the robot knew it. The evidence for the salesperson is that Tweety's owners had just told him so. The other hypotheses are defaults, and are used indiscriminately (i.e., regardless of whether evidence for them exists).

So, explaining is the process of finding the relevant facts and prototypes to model an agent. The problem mentioned in section 1.2, how to prove that it is inconsistent to assume a given hypothesis, deals with explaining as opposed to predicting. Predicting is as simple as running a non-nested Theorist system, given that we have already specified which of the available facts and hypotheses are applicable. In the pet shop, given that the robot has decided on the relevant facts and hypotheses of the salesperson, it is easy to predict that the salesperson thinks Tweety flies.

The problems associated with explaining, however, are many. We have already established that the model of the agent must be consistent with the observations. But there are usually many models that satisfy the constraints. For example, if we observe that the salesperson thinks that Tweety *doesn't* fly, we may assume that the salesperson thinks that Tweety is a penguin, or an emu, has clipped wings, or whatever, as long as it leads to the conclusion that Tweety doesn't fly. We could call this problem to finding the true model, the *model induction problem.*

Obviously, any solution must be heuristic. One can find many approaches to the problem in the user-modeling literature. For example, Rich [1979] uses an approach called *stereotypes,* where elaborate hypotheses are made on the basis of a few self-descriptive words that the agent offers. The most difficult aspect of this approach is stating what kinds of things can be assumed. Our approach is more fine-grained, and hence even more difficult, since we are talking about knowledge and belief, whereas Rich only discussed broader features of the agent.

One promising approach is to directly *ask* the agent (if possible) what they believe. But, it makes little sense to ask the agent specifically whether they have a certain fact or default. They don't necessarily consciously operate in the Theorist fashion. All we can ask is what they believe, and then try to find a Theorist system to predict those beliefs. Doing so may enable us to decide among multiple theories.

Another thing we can do is to *tell* the agent things. Assuming they understand and believe us, we can incorporate this information into our model of them.

2.1. Interaction Between Facts and Defaults

Let's leave the model-induction problem for now, and consider how the rest of the algorithm would work, assuming that we can choose the proper elements to be in our model (to be chosen by an oracle, perhaps). First, we must better understand the semantics. In order to understand the semantics of Nested Theorist, we must decide what we want nested facts and defaults to mean. We consider all of the two-level

combinations.

Let's begin with the nested defaults, such as $\Delta_i\Delta_j p$. This syntax should mean that if consistent to do so, i will assume that, if consistent to do so, j assumes p. But when is it consistent for i to assume $\Delta_j p$? Is $\Delta_i\Delta_j\neg p$ inconsistent with it? As we discussed in section 1.2, no. We should allow i to make both assumptions $\Delta_j p$ and $\Delta_j\neg p$. In some cases j may use Δp, and in some cases $\Delta\neg p$. In the pet shop example, the default $f(X)\leftarrow b(X)$ was prefaced by $\Delta_r\Delta_e$.

To specify a clause as $F_i\Delta_j p$ means that the default $\Delta_j p$ must *always* be available. Again we see that there is no possibility of inconsistency at the outer level. The clause $F_i\Delta_j\neg p$, for example, is not inconsistent with it, since an agent can operate with Δp and $\Delta\neg p$ at the same time. (The agent is only required to not actually simultaneously *make* both assumptions. At the outer level, i acknowledges that conflicting defaults are *available* to j, not that both are *used* at the same time.) Note that the pet-shop example would have worked similarly if $f(X)\leftarrow b(X)$ had been prefaced by $F_r\Delta_e$. It is difficult to say which is more appropriate.

Clauses of the form $F_iF_j p$ *can* be inconsistent. $F_iF_j p$ is inconsistent with $F_iF_j\neg p$, for example. The clause is intended to mean that i always believes that j believes p. In the pet shop example, $b(X)\leftarrow e(X)$ was prefaced by F_rF_e, because we can view the clause as something that is always believed in the domain of birds. (Again, this use is arguable. Perhaps $F_r\Delta_e$ is more apt.)

Finally, consider clauses of the form $\Delta_iF_j p$. They are intended to mean that i may assume that j always believes p as a fact. (In the example, the clause $e(tweety)$ was treated this way.) It is inconsistent for i to assume both $F_j p$ and $F_j\neg p$. Both facts may be available in the form $\Delta_iF_j p$ and $\Delta_iF_j\neg p$, but only one at a time can be assumed by i. $\Delta_iF_j p$ and $F_iF_j\neg p$ are not mutually inconsistent either, but if the two exist together, the default $F_j p$ can never be used. (Note the difference between inconsistency and uselessness with respect to nested defaults. $F_iF_j p$ is inconsistent with $F_iF_j\neg p$; $\Delta_iF_j p$ is useless if $F_iF_j\neg p$ exists, but is consistent.)

In summary, the semantics of the F and Δ operators is that anything within the scope of F must always be used in an explanation, and anything within the scope of Δ is optional. We explain something by finding a subset of instances of the defaults which together with the facts explain the goal. This is just as in non-nested Theorist, except that now we must choose the subset of instances of defaults at every level, for every agent involved.

2.2. Algorithm

We can make this semantics clearer, and at the same time write a pseudo-Prolog program to implement it. We define a predicate *explain*(*Facts*∪*Defaults*, *Agents*, *Goal*) which is true if the *Facts* and *Defaults* together can explain the *Goal* that is nested within the list of *Agents*. Note that, for now, we make no distinction between prototypes and defaults. The algorithm/semantics follows:

$explain(FD, [A1|A], G) \leftarrow$
$\quad F_{A1}^{FD} = \{w \mid F_{A1}w \in FD\}$
$\quad \Delta_{A1}^{FD} = \{w \mid \Delta_{A1}w \in FD\}$
$\quad \delta \subseteq \Delta_{A1}^{FD}$
$\quad explain(F_{A1}^{FD} \cup \delta, A, G)$
$\quad \neg prove(F_{A1}^{FD} \cup \delta, A, false);$

$explain(FD, [A1], G) \leftarrow$
$\quad F_{A1}^{FD} = \{w \mid F_{A1}w \in FD\}$
$\quad \Delta_{A1}^{FD} = \{w \mid \Delta_{A1}w \in FD\}$
$\quad \delta \subseteq \Delta_{A1}^{FD}$
$\quad F_{A1}^{FD} \cup \delta \models G$
$\quad F_{A1}^{FD} \cup \delta$ *is consistent*;

Comparing this with the semantics of Theorist, we note many similarities. F and δ together imply G, where δ is a subset of the defaults. The main difference is that the only facts and defaults to be used imply G are ones with correct agent subscripts.

Viewing the above as an algorithm, we see that the second clause forms the base case of the recursion, where we have only one agent to consider. The first two lines of each clause form the chosen facts F_{A1}^{FD} and defaults Δ_{A1}^{FD}. This has the effect of stripping off one layer of defaults at each level of recursion. Then, as in the original Theorist, δ is chosen from the set of defaults, and is used to derive the goal.

The last line of each clause is a consistency check (written in two different ways). One thing we notice, from our explanation of the semantics above, is that things never actually need to be tested for consistency until the last level of recursion. For example, the clauses $\Delta_i F_j F_k p$ and $F_i \Delta_j F_k \neg p$ could be inconsistent if both of them are chosen as defaults, but we need only check this by testing at the last level whether p and $\neg p$ are inconsistent. We need not test whether $F_k \neg p$ and $F_k p$ are inconsistent, even though they are inconsistent. This is because anything inconsistent at higher levels will also be inconsistent at the lowest level. (If two clauses are consistent at the lowest level, no

added prefaces, F or Δ, can make them inconsistent.) Hence, the predicate "*prove*" doesn't need to be defined. All the consistency checking can be done directly in the base clause.

Let's run through the pet-shop example to see how the robot's view of the salesperson's view explains that Tweety doesn't fly. (We are considering the situation after the salesperson has been told that the bird is an emu.) We list the sets F_i^{FD}, Δ_i^{FD}, and δ for the two iterations. (See Figure 3.)

iteration	F_i^{FD}	Δ_i^{FD}
1	$\{F_s b(tweety);$	$\{\Delta_s(f(X) \leftarrow b(X));$
	$F_s(b(X) \leftarrow e(X));$	$F_s e(tweety)\}$
	$F_s(\neg f(X) \leftarrow e(X))\}$	
2	$\{b(tweety);$	$\{f(X) \leftarrow b(X);\}$
	$b(X) \leftarrow e(X);$	
	$\neg f(X) \leftarrow e(X);$	
	$e(tweety)\}$	

Deriving that Tweety doesn't Fly

Figure 3

In the example, we never need to eliminate any defaults from Δ_i^{FD} until the end. Hence, at each iteration, $\delta = \Delta_i^{FD}$, except the last time, when $\delta = \{\}$. The facts by themselves derive $f(tweety)$. The only default, $f(X) \leftarrow b(X)$, is inconsistent.

3. A More Complex Framework.

We have good reason to be dissatisfied with the preceding system. Consider the clause $\Delta_i(p \rightarrow F_j q)$. It is intended to mean that if i can explain p, i will assume that j holds q as a fact. There is a dependence among agents here that we are unable to reason about in our previous system. Previously, all we could do was to strip off one operator, F or Δ, and then partition the resulting clauses. Now we are faced with a clause that cannot be partitioned that easily.

Consider a contrived example. Let p be the proposition, "the student answered the subtraction problem incorrectly." Imagine that the system knows that the teacher has looked at the student's answer. Now, when the system looks at the answer, it will decide whether p is true or false. If p is true, it will assume that the teacher has as a fact that p is true, since the teacher is unlikely to make the same mistake as the student. That is, if p is true, the teacher knows p is true. We can model this assumption of the system's by $\Delta_{system}(p \rightarrow F_{teacher} p)$, exactly the form of the clause introduced above.

There is no obvious way of extending the previous framework into a more complex one that handles this new example. Perhaps we need to represent each agent's view of the world as a possible world, which has some relationship to other agents' views. (This idea is due to David Poole. See [Moore, 1984] for a similar approach to auto-epistemic reasoning.) For example, if in the possible world representing the system's view p is true, then in the world representing the system's view of the teacher, p must be a fact. This idea provides enough flexibility to talk about many kinds of dependencies between different views of the world. We have yet to work out the details.

A new problem we have to deal with is to decide whether any arbitrary statement should be allowed. Can we state, for example, $F_g F_h \Delta_i p \vee F_j q \wedge explain(FD, [ij], r) \leftarrow \Delta_k \neg p$? Does that make any sense? A clause as simple as $\Delta_i(q \leftarrow F_j p)$ already seems strange. It says that i assumes that it believes q if j has p as a fact. It is strange because we can't really prove something like $F_j p$, and so the clause is difficult to put to use.

How about negative information? Does it make sense to say i believes by default that j doesn't have p as a fact ($\Delta_i(\neg F_j p)$)? If so, there are more ways that assumptions at higher levels can be inconsistent. (We considered this in section 1.2.)

Can we state that a particular default is believed by i to be held by everybody ($F_i \Delta_X p$) ? Some way of quantifying over agents must be permitted so that we can state things more compactly (i.e., and not have to state the same default for each possible agent).

Given all these options, we may need some way of restricting our language so that we are left with a manageable and meaningful subset.

4. Further Research

There is obviously much that could be done if this path of research is followed. For example, we may want to allow multiple extensions. Moore [1983] suggested that if an agent's model has multiple extensions, an outside observer could only know that the agent believes the intersection of those extensions. We can do better by adding theory preference, so that to choose our model of an agent, we imagine what theory preference algorithm the agent is using.

Another path of research is to model non-ideal agents, as Konolige [1985] attempts. This would mean representing various kinds of limitations on an agents ability to reason correctly.

And, of course, we could always look for efficient ways to implement the theory, once we are satisfied that it is a good theory. For example, how can a theory which incorporates possible worlds be implemented?

Another good idea would be to find suitable applications. Then we could study how a particular world could be axiomatized. Can we find a clean distinction between defaults and prototypes in the world, and will our framework prove fruitful?

5. Acknowledgements

Thanks to David Poole for constantly proving that my intuition was wrong, and thanks to the anonymous reviewers who made a large contribution to the final form and content of this paper.

6. References

Konolige, Kurt, "Belief and Incompleteness," in *Formal Theories of the Common-Sense World,* ed. Jerry Hobbs, Ablex Publishing Co., Norwood, NJ, 1985.

McCarthy, John, "Applications of Circumscription to Formalizing Common-Sense Knowledge," *Artificial Intelligence,* vol. 28, no. 1, pp. 89-116, February, 1986.

Moore, Robert C., "Possible-world Semantics for Autoepistemic Logic", *AAAI Workshop on Nonmonotonic Reasoning,* pp. 344-354, New York, NY, October, 1984.

Moore, Robert C., "Semantical Considerations On Nonmonotonic Logic", *Artificial Intelligence,* vol. 25, no. 1, p. 75, January, 1985.

Moore, Robert C., Reasoning about Knowledge and Action, in *Formal Theories of the Common-Sense World,* ed. Jerry Hobbs, Ablex Publishing Co., Norwood, NJ, 1985.

Poole, David L., "Default Reasoning and Diagnosis as Theory Formation," Technical Report CS-86-08, University of Waterloo, Waterloo, Ontario, March, 1986.

Rich, Elaine, "User Modelling Via Stereotypes," *Cognitive Science,* vol. 3, pp. 329-354, 1979.

Retrieval and Microcode Synthesis of Register-Transfer Modules as Theory Formation

Mossaddeq Mahmood

CS 786 Report
VLSI Group
Department of Electrical Engineering
University of Waterloo, Canada
April 24, 1987

ABSTRACT

Application specific integrated circuits (ASIC) can be designed, using the tools for datapath and microcode synthesis, at the register-transfer (RT) level. The retrieval and microcode synthesis of ASIC circuits from a design database is being investigated. A prototype system has been developed using the Theorist Reasoning System. A relational database representation has been used to describe the structure of the RT-level modules. Theory formation is used to model the retrieval and microcode synthesis. Rules are being written to model the behavior of the hardware primitives. Preliminary results indicate that the knowledge representation can be used for retrieval, microcode synthesis, and functional simulation of simple RT-level designs.

1. Introduction

Over the last decade, the development of microelectronics has resulted in the integration of large numbers of electrical devices on a single silicon chip. This development has increased the potential for designing powerful digital systems (usually the CPU) on a single integrated circuit (IC). The recent architectures for Lisp and Prolog machines are also the result of this development. The design of large digital systems is a complex task and requires a team effort of designers in various levels of design abstraction: architectural, register-transfer, logic, circuit and geometrical level of representation.

Our research objective is to automate the retrieval and microcode synthesis of application specific IC (ASIC) designs from a design database described at RT-level. This tool can also be used for symbolic simulation of ASIC designs to verify that an initial design description satisfies the behavioral specification. The ASICs are intended to be used in telecommunication applications. For this project we are investigating the modeling of our design problem in the Theorist Reasoning System (TRS) [Pool 86a]. Poole [Pool 86b] illustrates the use of TRS in diagnostic reasoning for fault diagnosis of combinational circuits.

An RT-level language describes a hardware in two different aspect: structural and behavioral. The structure of a hardware (machine model) is described in terms of interconnection of various machine resources. The behavior of a hardware (computation model) is described algorithmically in terms of register assignments, machine operations, and conditional branches. A subset of a procedural language (e.g C or Pascal) can be used to model the behavioral part of a hardware while its structural part can be modeled using an object oriented representation.

The compiler of a hardware description language outputs an intermediate representation (IR) of the algorithm to the lower level tools. The IR is similar to the basic block and the flow graph concept used by optimizing compilers [Aho 86]. A data flow and control flow analysis performs machine independent optimization on the IR and provides necessary housekeeping information to lower level tools (e.g. datapath synthesis, microcode synthesis tools). The tasks at RT-level synthesis are operation placement, data routing and register assignment. These tasks are interdependent.

1.1. Design Synthesis

Design synthesis of digital systems is analogous to the planning problem with constraints, as described in [Stef 81], but of greater complexity. The constraints are primarily based on functionality, chip area, speed, power, design style and technology. Design at register-level includes synthesis of the data-path and the control-unit. There has been limited effort to automate synthesis in logic programming environment.

Finger and Genesereth [Fing 85] present the Residue approach for deductive design synthesis. The Residue approach is similar to the Theorist reasoning system. The designs are represented as sets of constraints ($=$ and $\neq$) and logical relations ($\wedge$, $=>$). The system finds a residue for a given design goal. The theorem prover uses constraint propagation of symbolic expressions to prune the search space. Residue uses a direct proof procedure rather than a refutation method and is written in the MRS logic programming language [Gene 84]. In this approach evolving designs are expressed as a set of propositions, rather than a single term approach (e.g. Prolog). Residue resolution has also been used in the DART program [Gene 84] for diagnostic reasoning.

de Kleer and Sussman [deKl 80] present SYN, a tool for analog circuit synthesis. It performs incremental deduction to solve the sub-problems that are solvable and propagates the solution to the remaining parts of the circuit. It uses symbolic constraint propagation when the sub-problem is unsolvable and uses an algebraic manipulator to reduce expressions into a canonical form. The time and space complexity of the symbolic manipulator often limits it to solving small designs. In both approaches the circuit configuration is fixed. Residue synthesizes the inputs of a circuit to a given output value. SYN synthesizes the input output relation as algebraic equations.

1.2. Retrieval of Design Modules

The VLSI designs are described at various levels of abstraction. The representation at the register-transfer level is complex due to the interaction of control signals, and data-path. As the size of the design database grows it becomes difficult to find a suitable module based on the functionality and design constraints. There are recent efforts to automate the storage and retrieval of VLSI designs.

Foo and Kobayashi [Foo 86] present NEPTUNE, a knowledge-based system for VLSI module selection. The frame knowledge representation scheme has been used to represent modules such as gates, controllers etc. The selection criteria are functionality, delay, area and power. The system first uses a heuristic search to find a list of instances which satisfy the subfunctions and design constraints. Next a dependency directed

backtracking algorithm evaluates the designs with respect to design constraints. Finally, the modules are ordered using a rank-and-sort strategy based on the evaluation results. NEPTUNE is coded in the language C.

Ho et. al. [HoHu 85] also proposed a frame-based storage and retrieval system for VLSI cells. The system uses default, heuristic and logical reasoning for search and consistency checking.

2. Register-Transfer Level Specification

The retrieval and microcode synthesis is based on SDC notation, an RT-level design specification proposed in [Mava 84]. The SDC model has three basic primitives: selectors, delays and combinationals. The selector has two data inputs. The control input to the selector selects one of the data inputs as the value of the output. The output of a delay element lags its input by one time unit. The data output of a combinational element is expressed in functional notation as $f(x_1, .. x_n)$, where $x_1 .. x_n$ are the inputs. Figure 1 illustrates these three primitives and their algebraic relations.

A circuit described in SDC notation is represented as an interconnection of the three primitives. The schematic of a simple read-write register in SDC notation is shown in Figure 2. The RT-level model of a module consists of a data-path and a control-unit. The data-path has a set of data-inputs, data-outputs, selector inputs and status outputs. The data-path is formed by the interconnection of selectors, delays and combinationals. The module communicates to the environment through the input and output ports. Figure 3 illustrates the schematic representation of an RT-module. The status-outputs of the data-path are connected to the status inputs of the control-unit. Similarly, the action-outputs of the control-unit are connected to the path selector inputs of the data path. The RT-module data path is programmable and can perform different computations depending on the microcode and state transition of the control-unit. The program is represented as a table with entries for control-inputs, status-inputs and action-outputs. A digital system is represented as the interconnection of a number of modules.

The behavior of an RT-module is described using a simple language. Figures 4a and 4b illustrate the syntax of the input specification and sample descriptions respectively. The specification language is intended to represent ASIC designs, where computation can be performed in parallel (similar to horizontal microprogramming designs). For example, the computation of *Exp1* and *Exp2* of an *if (Cond Exp1 Exp2)* statement can be evaluated in parallel if the hardware is available in the design. The control-unit selects one of the expressions depending on the status feedback from the *Cond* expression.

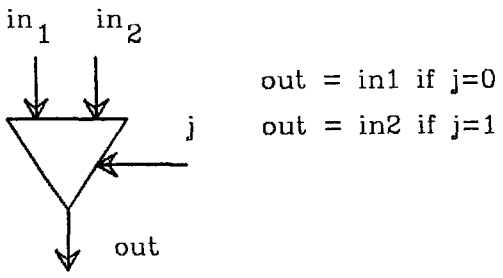


Figure 1a: A Selector Primitive.

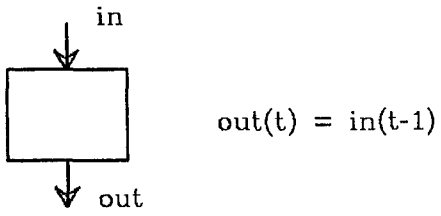


Figure 1b: A Delay Primitive.

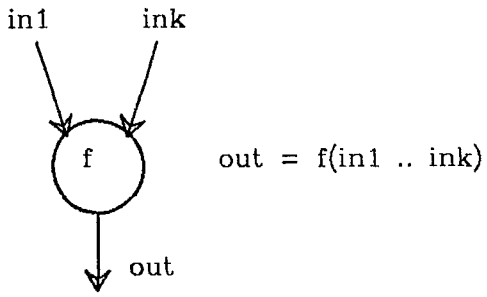


Figure 1c: A Combinational Primitive.

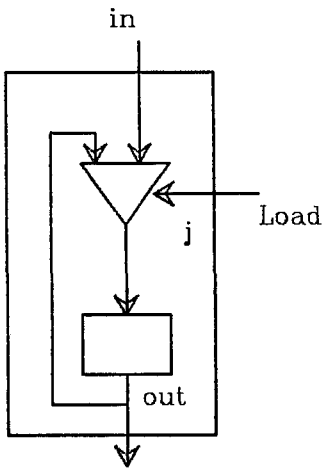


Figure 2: A Read-Write Register.

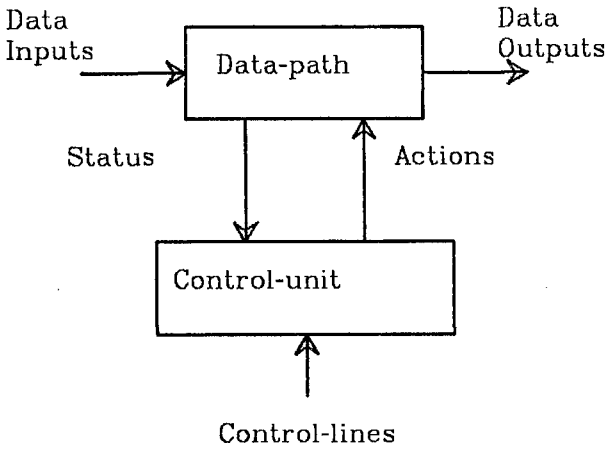


Figure 3: A Register-Transfer level Module.

```
Exp ::= if (Cond Exp1 Exp2) | Arith_op (Exp1 Exp2) | Data-In
Arith_Exp ::= Arith_op (A_Exp1 A_Exp2) | Data-In
Cond ::= Relat_op (A_Exp1 A_Exp2)
Exp1, Exp2 ::= Exp
A_Exp1, A_Exp2 ::= Arith_Exp

Relat_op ::= gt | lt | le | ge | eq | ne
Arith_op ::= plus | minus | mul
Data-In ::= a | b | c | ....
```

Figure 4a: Syntax of the input specification.

```
% Functional representation
  out = if (gt(a, b), plus (x, minus (y, z)), x)

% Procedural interpretation
  if a > b then
      out := x + y - z
        else
      out := x
  end_if
```

Figure 4b: A sample input specification.

We are looking at the following design problems:

1. Automate the retrieval of designs from database.
2. Automate the generation of microcode (or control-table) for the control-unit.

It is also possible to perform symbolic simulation to verify that a design meets the behavioral specification during initial stage of the design process.

Previous effort was concentrated in the area of microcode synthesis in the Prolog environment [Nel 84, Bala 85, Mahm 86]. These various versions derive the necessary control signals to implement a function in a fixed circuit. The problems with previous approaches were:

1. The knowledge representation was inappropriate.
2. The use of side effects [Bala 85, Mahm 86].

The system [Bala 85, Mahm 86] was capable of solving complex problems but it was not suitable for further extension due to a non logical knowledge representation.

3. Knowledge Representation in TRS

The knowledge base of the Theorist Reasoning System consists of a collection of logical formulas classified as possible hypotheses (Δ), facts (F), and observations (G). "The Theorist reasoning strategy attempts to accumulate consistent sets of facts and instances of hypotheses as explanations for which the observations are logical consequences " [Pool 86a p. 2].

The *Facts* database contains the design descriptions of the RT-level modules and the rules describing the SDC primitives. This representation scheme is similar to the DART program [Gene 84]. Figure 5 illustrates the structural description of circuit *f2* (see Figure 6). The structure of the RT-level modules is represented as an interconnectional relation along the data-axis and control-axis. Other relations are *module, part_of, in_port,* and *out_port*. For example, the proposition (relation)

data_conn (f2, in(1 f2), in(1 add1))

specifies that the input port *1* of device (module) *f2* is connected to the input port *1* of device *add1* along the data-axis.

```
fact module (f2);
fact out_port (f2 1);
fact in_port (f2 1);
fact in_port (f2 2);
fact in_port (f2 3);
fact part_of (f2 add1 plus);
fact part_of (f2 sel1 selector);
fact part_of (f2 d1 delay);
fact part_of (f2 d2 delay);
fact data_conn (f2 in(1 f2) in(1 add1));
fact data_conn (f2 in(2 f2) in(2 add1));
fact data_conn (f2 in(3 f2) in(1 d1));
fact data_conn (f2 out(1 add1) in(1 d2));
fact data_conn (f2 out(1 d2) in(1 sel1));
fact data_conn (f2 out(1 d1) in(2 sel1));
fact data_conn (f2 out(1 sel1) out(1 f2));
fact control_conn (f2 in(1 f2) in(1 sel1));
```

Figure 5: Facts describing the structure of Module f2.

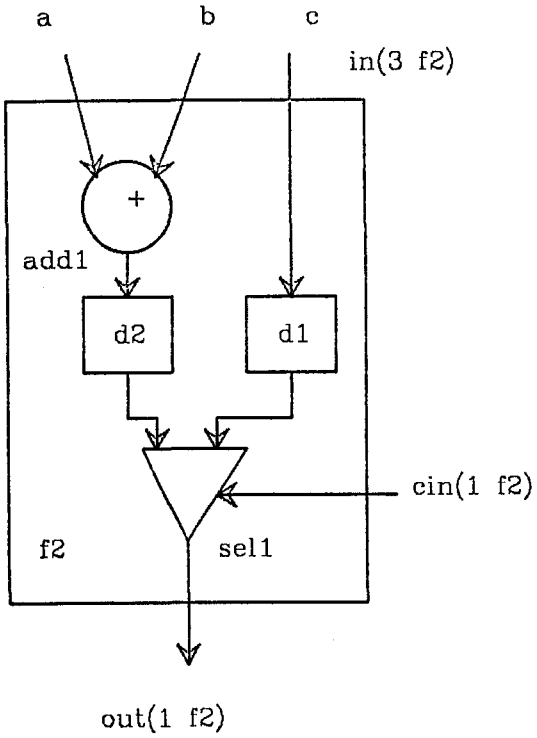


Figure 6: Circuit description of Module f2.

The output behavior of an operator is described in terms of its input arguments. For example, the proposition

operator (plus (In1 In2), plus, [In2 In1], arith, 2)

specifies that the output of arithmetic operator *plus* is *plus (In1 In2)*, if the two inputs are *In1* and *In2*, where *In1* and *In2* are two functional expressions. Figure 7 illustrates the specification of the relational and arithmetic operators.

```
% Relational Operators
fact operator (gt(In1 In2) gt [In2 In1] cond 2);
fact operator (lt(In1 In2) lt [In2 In1] cond 2);
fact operator (le(In1 In2) le [In2 In1] cond 2);
fact operator (ge(In1 In2) ge [In2 In1] cond 2);
fact operator (eq(In1 In2) eq [In2 In1] cond 2);
fact operator (ne(In1 In2) ne [In2 In1] cond 2);

% Arithmetic Operators
fact operator (plus(In1 In2) plus [In2 In1] arith 2);
fact operator (minus(In1 In2) minus [In2 In1] arith 2);
fact operator (mul(In1 In2) mul [In2 In1] arith 2);
```

Figure 7: Rules for relational and arithmetic operators.

A set of complex rules specify the output behavior of a port of a module in terms of the interconnection of primitives, selector inputs, status output and data-inputs. Figure 8 illustrates the rules used to model the SDC primitives. The rule for a selector specifies that the value at output port *1* of device *D* of module *M* is *Out* at clock pulse *Clk*, if the body of the clause can be proven. The set of rules describing the input output relation of the hardware elements attempt to propagate the behavioral expression (or subexpressions) through the network of elements from the output port of the module to its inputs along the data-axis (backward chaining). The proposition *value(1 a 0)* specifies that the input of port *1* of any module in the database is *a* at clock *0*. While propagating the behavioral expression, the consistent set of defaults (*code* and *status*) is generated by the TRS system.

Our goal is to observe a certain functional behavior *Value* at the output port *PortNo* of an RT-module *M* in the database. We represent this goal as

function (M, PortNo, Value).

We want the TRS to retrieve a module from the database and generate the entries of

```
% rule for input ports of a module.
rule data (in(P M) M V T) <-
        module (M)
        in_port (M P)
        val(V P T);

% rule for connection along data-axis
rule data (Y M Val Clk) <-
        data_conn (M X Y)
        data (X M Val Clk);

% rule for if (Cond Exp1 Exp2) statement
rule data (out(1 D) M if(C E1 E2) Clk) <-
        part_of (M D selector)
        operator (C Oper T cond N)
        part_of (M NewD Oper)
        control_conn (M out(Y M) out(X NewD))
        data_all (T N M NewD Clk1)
        plus (1 Clk1 Clk)
        status_code (Status Port Code)
        pick (Port [E1 E2] NewE)
        control_conn (M in(X1 M) in(P1 D))
        data (in(Port D) M NewE Clk)
        cond_status (cout(Y) Status Clk)    % hypothesize status
        micro_code (cin(X1) Code Clk);      % hypothesize code

% Rule for a selector element.
rule data (out(1 D) M Out Clk) <-
        part_of (M D selector)              % D is of type selector
        selector_code (Code Port)           % Code for selecting Port
        control_conn (M in(X M) in(Y D))    % connection along control-axis
        data (in(Port D) M Out Clk)         % Recursive call
        micro_code (cin(X) Code Clk);       % Generates theory: code (X Y Z)

% rule for arithmetic operator
rule data (out(1 D) M Out Clk) <-
        operator (Out Oper T arith N)
        part_of (M D Oper)
        data_all (T N M D Clk) ;

% rule for a delay element
rule data (out(1 D) M Out Clk) <-
        part_of (M D delay)
        data (in(1 D) M Out Clk1)
        plus (1 Clk1 Clk) ;
```

Figure 8: Rules for SDC primitives.

the control-table. The hypotheses (or defaults) i.e. the *code, status,* and *retrieved-module* model the various results (entries to the control-table) required to explain the goal (observation). Figure 9 illustrates the logical formulas for the set of hypotheses used for the RT-level designs. The TRS generates a consistent *theory* to explain the desired goal. The *theory* consists of ground instances of the hypotheses. The default

retrieved-module (out(PortNo, M), clk(Clock), val(Value))

represents the *Value* observed at the output port *PortNo*, of an RT-module *M* valid at clock pulse *Clock*.

If the module can explain the behavior at its output, then the sequence of microcode (Actions) and status inputs for the control-unit (see Figure 3) are generated. The default *code (cin(1), on, 2)* specifies that the *code* for control-input port *1* of the retrieved module is *on* at clock pulse *2*. The default *status (cout(2), true, 1)* specifies that the *status* for control-output port *2* of the retrieved module is *true* at clock pulse *1*. The values {*on, off*} for *code* and {*true, false*} for *status* are the required entries to the control-table. A blank entry means a don't-care condition. The clock pulse sequence represents the state transition for the control-unit.

```
% retrieved-module (out(Port Module), clk(Clock), val(Value));
default retrieved-module (out(PortNo M), clk(Clk), val(Value)) :
      function (M, PortNo, Value) <-
         module (M)              % M is a module
         out_port (M, PortNo)    % PortNo is an output port of M
         data (out(PortNo M), M, Value, Clk);

% code (cin(Port) Code Clock);
default code (X Y Z) : micro_code(X Y Z) <-;

% status (cout(Port) Status Clock);
default status (X Y Z) : cond_status (X Y Z) <-;
```

Figure 9: Set of Hypotheses (defaults) used for SDC-based RT-level designs.

The TRS system performs incremental consistency checking of a default by attempting to prove its negation. If the negation of the default can be proven then the default is inconsistent. Figure 10 shows the constraint rules used in our model to eliminate inconsistent defaults. For example, the control input to a port cannot be *on* and *off* during the same time interval. The fact

n(code(cin(1) on 1)) <- code (cin(1) off 1) ne (on off)

can be proven during incremental consistency checking of a default. The TRS system discards inconsistent defaults.

```
% During incremental consistency checking of ground instances of
% defaults the TRS system attempts to prove the negation of the
% default (non-monotonic reasoning).
% The following facts represent the rules to prove negative instances.
% If the proof is successful then the default is rejected.

fact n(code (Dev Y Clk))  <- code (Dev Y1 Clk) ne (Y Y1);
fact n(status (Dev Y Clk))  <- status (Dev Y1 Clk) ne (Y Y1);
fact n(micro_code (Dev Y Clk))  <- micro_code (Dev Y1 Clk) ne (Y Y1);
fact n(cond_status (Dev Y Clk))  <- cond_status (Dev Y1 Clk) ne (Y Y1);
```

Figure 10: Constraint rules to eliminate contradictory defaults.

4. Preliminary Results

We have considered a database containing two simple designs that should demonstrate the objective of this project. Appendix 1 describes the logical and graphical descriptions of the circuits. Figure 11 shows a Theorist session.

The query : *explain function (f2 1 plus(a b))* attempts to explain the value of *plus(a b)* at output port *1* of module *f2*. If required, the *code* (selector inputs) and *status* outputs are synthesized to *explain* the observation. The query : *explain function (M P plus(a b))* attempts to find a module *M* which can *explain* the functional behavior at output port *P*. The modeling is based on logical deduction, therefore we can perform symbolic simulation of the RT-level designs. We can also *explain* multiple observations. The query : *explain function (f2 1 plus(a b)) code (cin(1) off 1)*; attempts to prove that the module *f2* can *explain* the functional behavior at port *1* with the given *code*.

The *if (Cond, Exp1, Exp2)* expression used by the model is different compared to a sequential computation. The conditions of a nested *if* can be evaluated and tested in parallel. To model all the mutually exclusive selections depending on the *status* feedback, we would require multiple explanations in the TRS. Each explanation models a row of the control-table. Table 1 shows the control-table entries for a nested *if* expression. The second query in Figure 11 shows the query posed to the TRS and the multiple explanations generated by Theorist. The blank represents a don't-care condition.

```
Welcome to Theorist 0.3 compiler
For help type "h;"
All complaints to David Poole (dlpoole@watdragon)

: consult synthesis;

: explain function(f2 P X);

yes
answer:[function(f2,1,plus(a,b))]
theory:retrieved-module(out(1,f2),clk(1),val(plus(a,b)))
       code(cin(1),off,1)

yes
answer:[function(f2,1,c)]
theory:retrieved-module(out(1,f2),clk(1),val(c))
       code(cin(1),on,1)

no (more) answers

: explain function(M P if(gt(a b) if(lt(b c) plus(a b) minus(a b)) mul(a b)));

yes
answer:[function(f1,1,if(gt(a,b),if(lt(b,c),plus(a,b),minus(a,b)),mul(a,b)))]
theory:retrieved-module(out(1,f1),clk(1),
val(if(gt(a,b),if(lt(b,c),plus(a,b),minus(a,b)),mul(a,b))))
code(cin(2),off,1) status(cout(1),true,1)
code(cin(1),off,1) status(cout(2),true,1)

yes
answer:[function(f1,1,if(gt(a,b),if(lt(b,c),plus(a,b),minus(a,b)),mul(a,b)))]
theory:retrieved-module(out(1,f1),clk(1),
val(if(gt(a,b),if(lt(b,c),plus(a,b),minus(a,b)),mul(a,b))))
code(cin(2),off,1) status(cout(1),true,1)
code(cin(1),on,1) status(cout(2),false,1)

yes
answer:[function(f1,1,if(gt(a,b),if(lt(b,c),plus(a,b),minus(a,b)),mul(a,b)))]
theory:retrieved-module(out(1,f1),clk(1),
val(if(gt(a,b),if(lt(b,c),plus(a,b),minus(a,b)),mul(a,b))))
code(cin(2),on,1) status(cout(1),false,1)

no (more) answers

: explain function(F P minus(a b));

yes
answer:[function(f1,1,minus(a,b))]
theory:retrieved-module(out(1,f1),clk(1),val(minus(a,b)))
code(cin(2),off,1) code(cin(1),on,1)

no (more) answers
```

Figure 11: A Theorist session for retrieval and microcode synthesis.

Functional Query: if(gt(a b), if(lt(b c), plus(a b), minus(a b)), mul(a b)).
Retrieved Module: f1
Value at Output Port: 1
Valid at clock pulse: 1

Time	Status-Out		Control-In	
	cout(1)	cout(2)	cin(1)	cin(2)
0				
1	true	true	off	off
0				
1	true	false	on	off
0				
1	false			on

Table 1: Control-table representing the selector *code* and *status* feedback for module *f1*.

The source code for the prototype system is documented in the ~u/mmahmood/synthesis file. A sample session using the Theorist compiler (version 0.3) is available in the ~u/mmahmood/.results file.

5. Future Work and Conclusion

In this project, we have demonstrated the use of the Theorist Reasoning System in retrieval and microcode synthesis of register-transfer level designs. At this moment, we feel that the inclusion of the following features will enhance the retrieval and microcode synthesis of RT-level designs.

1. Addition of rules to represent feedback paths (loops) in the circuit. The loop iteration has to be constrained due to the depth-first traversal of the circuit (otherwise it will loop infinitely).

2. Addition of a simple canonical symbolic manipulator to aid in retrieving modules that are functionally equivalent to the input specification.

3. The current prototype searches sequentially through the module database during module retrieval. Heuristic knowledge can be added to prune the search space based on the functionality and style of the designs.

4. Addition of other selection criteria such as area, power, etc. It is possible to collect the instances of modules which satisfy the design criteria before performing the depth-first traversal of the circuit. Finally, we can also sort the instances of modules based on the order of priorities of design constraints.

5. Addition of other control constructs (*if_branch (Cond Block1 Block2)*), shift operations, etc. to represent a wider class of designs. This will require control flow analysis of the input specification, represented as flow graphs [Aho 86].

The tasks at RT-level synthesis are operation placement, data routing and register assignment. These tasks are interdependent. The current prototype use the depth first search strategy to traverse a module. The rules for feedback will add another dimension to the complexity of the search. The number of modules in the database and the chronological backtracking of Prolog will also contribute to the complexity. The main problem with the current modeling is that the rules are modeled for general topology which requires a backtracking control structure. The backtracking program will always create problem due to complex interaction of the three interdependent task at RT-level. The interaction is both in space and time, and requires global housekeeping. A reasonable way to reduce the combinatorial explosion is to restrict the RT-modules to known design styles and use non-backtracking algorithms.

The current version of Theorist (version 0.3) is extremely slow. This is due to the inefficient backtracking scheme used by Prolog and the semidecidable consistency checking algorithm. There are two areas, dependency directed backtracking, and symbolic constraint propagation can provide new concepts toward improving the search strategy in the TRS. A dependency directed backtracking can improve the efficiency of search (consistency checking can use the partially computed results). Symbolic constraint propagation has been used in a number of synthesis tools [Gene 84, deKl 80, Stef 81] to prune the enumerative search space.

In this project we have used the Theorist Reasoning System for retrieval and microcode synthesis of simple RT-level modules. We hope that the proposed enhancements to the existing prototype and the TRS will provide a new concept for solving design tasks at RT-level.

Acknowledgements

I would like to thank Scott D. Goodwin for his suggestion about the modeling of the problem in the TRS system. I would also like to thank the students of the winter-87 CS786 class for their helpful advice - particularly Paul Strooper, Amar Shan and the reviewers of this paper. I am also indebted to Randy Goebel, David Poole, Farhad Mavaddat and M. I Elmasry who have greatly influenced my work.

References

[Aho 86] A.V. Aho, R. Sethi, and J.D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley 1986.

[Bala 85] R. Balaz, Work-Report, University of Waterloo, Fall 1985.

[deKl 80] J. de Kleer and G.J. Sussman, "Propagation of Constraints applied to Circuit Synthesis", International Journal of Circuit Theory and Applications. Vol. 8, 1980, pp. 127-144.

[Fing 85] J.J. Finger and M.R. Genesereth, "RESIDUE: A Deductive Approach to Design Synthesis", Report No STAN-CS-85-1035, Department of Computer Science, Stanford University, January 1985.

[Foo 86] Y.S. Foo and H. Kobayashi, "A Knowledge-Based System for VLSI Module Selection", ICCD 86, pp. 184-187, 1986.

[Gene 84] M. R. Genesereth, "The use of Design Descriptions in Automated Diagnosis", Artificial Intelligence, Vol. 24, 1984, pp 411-436.

[HoHu 85] W.P.-C. Ho, Y.H. Hu, and D.Y.Y. Yun, "An Intelligent Librarian for VLSI Cell Databases", Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers", 1985, pp. 78-81.

[Mahm 86] M. Mahmood, Project Report for CS 686, University of Waterloo, April 1986.

[Mava 84] F. Mavaddat, "A Model for Register-Transfer Level Design Specification: The *SDC* Notation", Technical Report CS-84-34, University of Waterloo, Oct. 1984.

[Nel 84] L.D. Nel, "Use of *SDC* notation in a Logic Programming environment", University of Waterloo, Sept. 1984.

[Pool 86a] D. Poole, R. Goebel and R. Aleliunas, "Theorist: a Logical Reasoning System for Defaults and Diagnosis", Technical Report CS-86-06, Faculty of Mathematics, University of Waterloo, February, 1986.

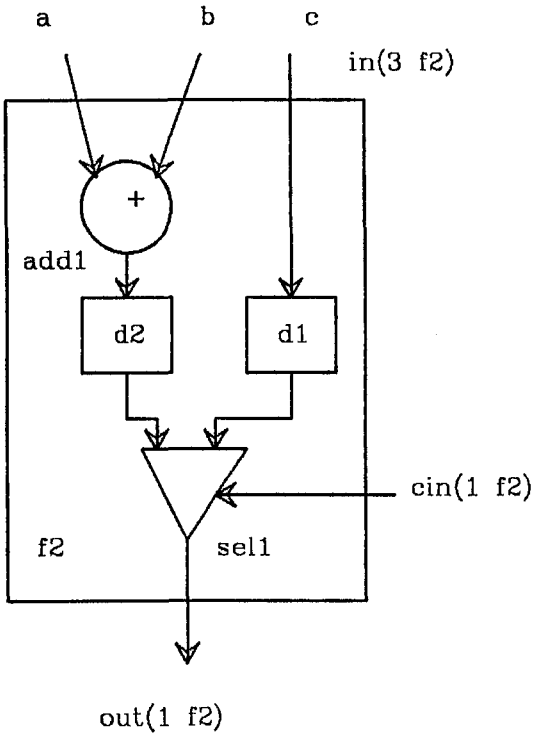
[Pool 86b] D. Poole, "Default Reasoning and Diagnosis as Theory Formation", Technical Report CS-86-08, Faculty of Mathematics, University of Waterloo, March, 1986.

[Stef 81] M. Stefik, "Planning with constraints", Artificial Intelligence, Vol. 16, 1981, pp. 111-140.

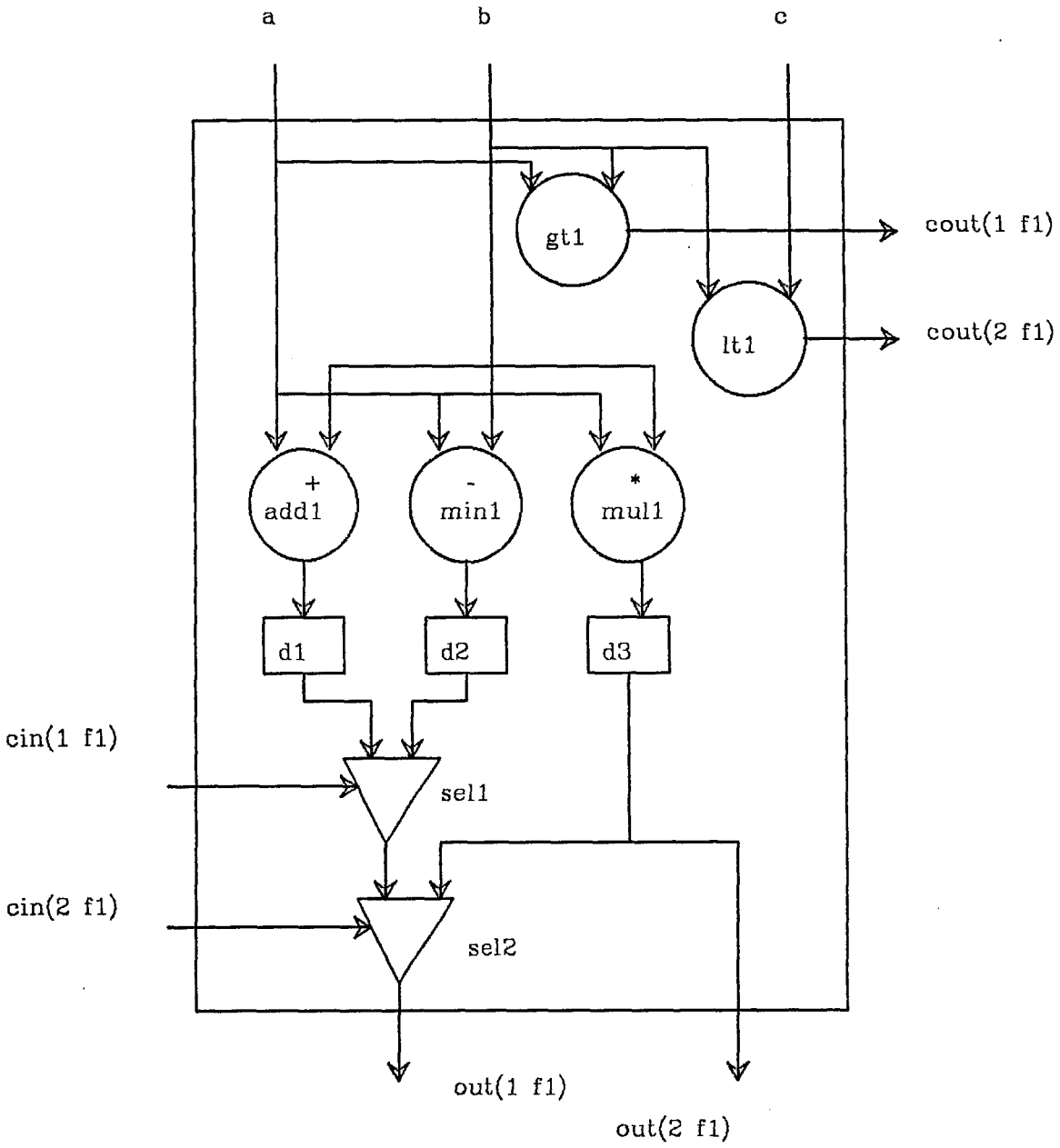
Appendix 1

```
fact module (f2);
fact out_port (f2 1);
fact in_port (f2 1);
fact in_port (f2 2);
fact in_port (f2 3);
fact part_of (f2 add1 plus);
fact part_of (f2 sel1 selector);
fact part_of (f2 d1 delay);
fact part_of (f2 d2 delay);
fact data_conn (f2 in(1 f2) in(1 add1));
fact data_conn (f2 in(2 f2) in(2 add1));
fact data_conn (f2 in(3 f2) in(1 d1));
fact data_conn (f2 out(1 add1) in(1 d2));
fact data_conn (f2 out(1 d2) in(1 sel1));
fact data_conn (f2 out(1 d1) in(2 sel1));
fact data_conn (f2 out(1 sel1) out(1 f2));
fact control_conn (f2 in(1 f2) in(1 sel1));

fact module (f1);
fact out_port (f1 1);
fact out_port (f1 2);
fact in_port (f1 1);
fact in_port (f1 2);
fact in_port (f1 3);
fact part_of (f1 add1 plus);
fact part_of (f1 min1 minus);
fact part_of (f1 mul1 mul);
fact part_of (f1 sel1 selector);
fact part_of (f1 sel2 selector);
fact part_of (f1 gt1 gt);
fact part_of (f1 lt1 lt);
fact part_of (f1 d1 delay);
fact part_of (f1 d2 delay);
fact part_of (f1 d3 delay);
```

```
fact control_conn (f1 in(1 f1) in(1 sel1));
fact control_conn (f1 in(2 f1) in(1 sel2));
fact control_conn (f1 out(1 f1) out(1 gt1));
fact control_conn (f1 out(2 f1) out(1 lt1));
fact data_conn (f1 in(1 f1) in(1 add1));
fact data_conn (f1 in(2 f1) in(2 add1));
fact data_conn (f1 in(1 f1) in(1 min1));
fact data_conn (f1 in(2 f1) in(2 min1));
fact data_conn (f1 in(1 f1) in(1 mul1));
fact data_conn (f1 in(2 f1) in(2 mul1));
fact data_conn (f1 in(1 f1) in(1 gt1));
fact data_conn (f1 in(2 f1) in(2 gt1));
fact data_conn (f1 in(2 f1) in(1 lt1));
fact data_conn (f1 in(3 f1) in(2 lt1));
fact data_conn (f1 out(1 add1) in(1 d1));
fact data_conn (f1 out(1 min1) in(1 d2));
fact data_conn (f1 out(1 mul1) in(1 d3));
fact data_conn (f1 out(1 d1) in(1 sel1));
fact data_conn (f1 out(1 d2) in(2 sel1));
fact data_conn (f1 out(1 sel1) in(1 sel2));
fact data_conn (f1 out(1 d3) in(2 sel2));
fact data_conn (f1 out(1 sel2) out(1 f1));
fact data_conn (f1 out(1 d3) out(2 f1));
```



rcuit description of Module f2



Circuit description of Module f1

The Project for the Course CS786

A Pronoun Resolution System Based on Theorist Framework

Prepared by Fei Song
ID Number: 86 903 923

Professor: David Poole

April 20, 1987

ABSTRACT

A new pronoun resolution system is presented, which uses not only discourse structure, especially the theme and the focus of the discourse, but also constraint and default rules about lexicon, syntax, semantics and discourse to help resolve pronoun references. The system consists of two levels of control: Meta-control and Theory Formation/Preference. The former can decide the change of the themes in the discourse and manage the focus set in the Database. It also uses discourse-level of defaults to control the pronoun resolution over the Theory Formation/Preference. The latter is based on the Theorist Framework, which takes the constraints and defaults from the lexical level to semantic level as the hypotheses set and generates a group of consistent theories by using default reasoning. The preference of these theories is made by using a natural measure, the number of rules in a theory, whose capability is shown by a number of examples, especially for three cases: exceptions to constraints, some ambiguous references and ill-formed sentences.

1. Introduction

Reference resolution is one of the hardest problems in natural language understanding. It is essentially that of how words are able to denote concepts, and in particular how a certain sequence of words can denote a unique concept. That is, we need to identify concepts when they are initially referenced and to identify subsequent references to them. The reference to some concept or entity is called an anaphor and the referenced concept or entity is called the referent or antecedent of the anaphor. Usually we consider the reference resolution in a discourse, by which we mean a section of text, either written or spoken, which is coherent in the sense that it forms a unified whole.

There are two major kinds of referring (see [2]). The first is an internal reference between a noun phrase and some pre-existing database object which represents a real world entity. In the figure 1 below, internal reference links the noun phrase NP1 "Jimmy Carter" to a representation of Jimmy Carter (who is described as the previous president of the US, etc). The other kind of referring is co-reference, which links a noun phrase to another noun phrase. The two noun phrases are said to co-refer, and both internally refer to the same database object, thus the same real world entity. In figure 1, the dashed link from NP2 "Jimmy" to NP1 is a co-reference link. The dot-dash link from NP2 to the database object is a virtual internal reference link which results from the co-reference link from NP2 to NP1 and from the internal reference link from NP1 to the database object. Internal reference and co-reference links are distinguished here because co-reference links can be established more easily using discourse context and will be discussed in detail later in this paper.

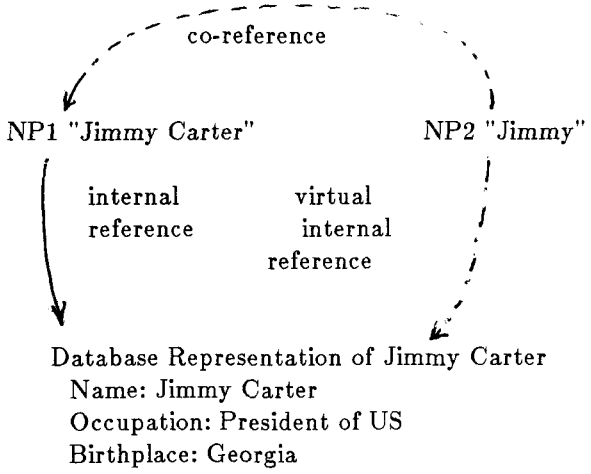


Figure 1: Reference Links Between Noun Phrases

There are also many types of anaphors, which can be summarized in the figure 2 below (see [5]).

Type of anaphor	Lexical realization
Pronominal	
.pronouns	he, she, it, they, one, ...
.epithets	the idiot, that stinking lump of camel excrement, ...
.surface count	the former, the latter, same, low ordinals, ...

Prosentential	it, so, ...
Proverbial	do
Proactional	do so, do it
Proadjective/prorelative	such, so, ...
Temporal	then, temporal relations
Locative	there, locative relations
Ellipsis	ϕ(empty)

Figure 2. The types of anaphors

Among these anaphors, pronouns are most frequently used and more difficult to resolve than other anaphors. They are usually marked for gender and/or number, which is often useful in resolution. For example,

1-1. Nadia wanted a gold ring, but Ross bought *her* a plastic one.

1-2. Ross bought { a radiometer | three kilograms of after-dinner mints } and gave { *it* | *them* } to Nadia for her birthday.

However there are some exceptions.

1-3. How is Swiss ? *She* is beautiful country.

1-4. The author thanks the reader for *their* kind indulgence.

In this paper, we will restrict us to the co-reference resolution of pronouns. Specifically, we will review some important ideas in the modern approaches to pronoun resolution in section 2, and summarize various kinds of defaults in pronoun resolution in section 3, and then in section 4, present a new pronoun resolution system based on Theorist Framework. In section 5, we will discuss more about theory formation and theory preference and last in section 6 give summaries and suggestions.

2. Some Important Ideas In Modern Approaches To Pronoun Resolution

There are two classes of approaches to pronoun resolution: traditional and modern (see [5]). The traditional approaches tend not to recognize as a separate problem the question of what is or isn't in consciousness. By consciousness, we imply the assumption that the speaker assumes the hearer has enough real-world knowledge in common with the speaker to know the entities which the speaker refers to, and that knowledge is what the speaker draws on in constructing a message for a hearer. Rather the traditional approaches assume that other things being equal, the set of possible referents is exactly the set of NPs(or whatever), from the whole of the preceding text, in strict order of recency. Their resolution methods tend to work at the sentence level, and may bring to bear world knowledge and low-level linguistic knowledge. Antecedents not explicit in the text are not handled.

On the other hand, modern approaches recognize the importance of focus and discourse-level knowledge for resolution. Implicit antecedents may also be handled. Intuitively, discourse organization usually centers around the theme, which is the main entity or concept that the discourse is about --- the subject central to the ideas expressed in the text. In the following example:

2-1. The boy is riding the horse.

The boy is clearly the theme and is riding the horse is a comment about the theme. However this is not the case in all contexts. If (2-1) were the answer to (2-2),

2-2. Who is riding the horse?

then the boy would be the comment and riding the horse the theme. In general, the choice of deciding the theme of a discourse is a difficult task. Much work has been done in attempting to capture precisely the concept of the theme and attempting to determine rules for deciding what the theme of a given text is.

Actually we can further distinguish themes into local theme, which refers to what a sentence is about, and global theme, which refers to what a discourse is about at a given point. These two concepts often coincide, but frequently don't. For example, in (2-3):

2-3. Nadia's chinchilla is shaped like a pear with a brush for a tail. *Its* teeth are long, but not very sharp.

The local and global theme of the first sentence are both Nadia's chinchilla. In the second sentence the global theme is unchanged from the first sentence, while the local theme is now Nadia's chinchilla's teeth.

Another important idea in modern approaches to pronoun resolution is the focus, by which we mean all concepts and entities from the preceding text that are referable at that point. It is quite similar to what we have called "consciousness". In another word, the focus indicates where to look for the antecedents for pronouns. It is necessary since without it, the number of possible referents grows with the length of the text. However the focus of a discourse alone is not sufficient to decide the referent. It must be used in conjunction with a hierarchical semantic network of associations. This network will indicate what other concepts are related to the focus. The network is a dynamic structure because the hearer adds to his/her general knowledge in the process of interpreting a piece of discourse. Also focus must include inferring mechanism to infer from general knowledge and some suppositions that a certain proposition is true. Here an example will be helpful to understand the focus (see [2]). in the discourse below, the focus of discussion is the meeting of sentence (2-4-1):

2-4-1. I want to schedule a meeting with Ira.

2-4-2. It should be at 3 p.m.

2-4-3. We can get together in his office.

2-4-4. Invite John to come, too.

All four sentences give information about the focus "meeting", but both sentence 3 and 4 make no direct reference to the meeting of sentence 1. As human hearer, we know that these sentences are related to the rest of the discourse because they give information about the focus meeting. In sentence 3, there are three clues which connect this sentence and the rest of the discourse: the use of get together, the co-reference of we to the participants of the meeting, and his office establishing a place for meeting. Sentence 4 introduces an additional participant which can be surmised from the use of invite, and the fact that ellipsis of the event that John is invited to is the focus. All of these should be done by making inferences on the general knowledge related to the focus.

From the discussion above, we can see that the modern approaches make the following assumption: given any point in a text, there is a set of focus associated with that point. Clearly the global theme should be included in the focus. Also as we move the reading of a discourse, some new local themes which are related to the global theme should be added to the focus, while some old local themes which are not related to the global theme too much should be eliminated from the focus. More formally if the point is a pronoun P, then there is a nominal focus set Fn, which consists of all those concepts that P could refer to for some following text. That is, Fn is a function of P and the preceding t defined by (see [5]):

$$Fn(t, P) = \{\ n \mid n \text{ is a NP in t, and } = t' \text{ such that } tPt' \text{ is well-formed English text in which P refers to n. } \}$$

At any given time, the nominal focus set Fn contains zero or more entities, which are possible referents for pronouns. When a pronoun needs resolving , one of several cases can occur:

(1) There is exactly one noun phrase in Fn, which fits the basic constraints in section 3; it is chosen as the referent.

(2) There are no suitable members of Fn; then the sentence with the alleged pronoun is probably ill-formed.

(3) There are more than one suitable members of Fn; then either a)we need to choose one of these possibilities, or b)the sentence is ambiguous.

Case 3 a) is the one of most interest here, since many apparent ambiguities of this kind can be resolved in our system.

3. Constraints And Defaults In Pronoun Resolution

In section 1, we have mentioned that pronouns are usually marked for gender and/or number. While these are strong constraints on reference, we saw in examples that they are not absolute: a plural pronoun can have a singular antecedent, a feminine one a neutral antecedent, and so forth. So constraints are a kind of defaults in the sense that they are "almost always" true, with a few exceptions.

There is also another kind of defaults which are usually weaker than constraints, but are quite useful in pronoun disambiguation. Many pronouns, like that of (3-1):

3-1. Ross told Daryel *he* had passed the exam.

are ambiguous --- he would be either Ross or Daryel. However, some which are theoretically ambiguous are in practice not:

3-2. Daryel told Ross *he(1)* was the ugliest person *he(2)* knew of.

In this example, each occurrence of the he could mean either Daryel and Ross, giving a total of four readings for the sentence. Yet most people immediately assume that he(1) is Ross and he(2) is Daryel without even noticing some or all of the other readings.

This indicates that in many cases of ambiguous pronouns, there is a preferred or default antecedent, which is taken as the correct one in the absence of contradicting context or knowledge. Hence we have the following preferred antecedent rule:"if you cannot decide on a single 'right' antecedent for the reference, choose from the uneliminated candidates the one that has quality X in the greatest proportion; if no candidate has significantly more of quality X than the others, treat the sentence as genuinely ambiguous". The quality X is called plausibility (see [5]).

In the example (3-2) above, the default interpretation is that Daryel is insulting Ross (he(1) = Ross, he(2) = Daryel), rather than being self-critical (he(1) = he(2) = Daryel). This may be simply because insulting behaviour is more common than openly self-critical behaviour with respect to personal appearance. That is, an insult is the most plausible interpretation of (3-2), and the corresponding antecedents are chosen accordingly.

Plausibility differs from other constraints mostly in its weakness. For example, the gender constraints that make (3-3) so bad:

3-3. Sue found *himself* pregnant.

are so strong that they are not really a matter of degree. Plausibility , on the other hand, is a matter of degree, and always requires evaluation relative to the other possibilities.

There are different levels of defaults which can be used in pronoun resolution:

(1) Lexical constraints: The example (3-4) below

3-4. Ross loves *his* wife and Daryel loves *his* wife too.

is ambiguous as to whose wife Daryel loves --- his own or Ross's. However (3-5) is not ambiguous in the same way the (3-4) is, simply because it is inherent in nose twitching that one can only do it

to one's own nose:

3-5. Nadia was able to twitch *her* nose and Ross was $\emptyset$ too.

Similarly, (3-6) is only two ways ambiguous and not four ways as is (3-2), since both pronouns must be co-referential:

3-6. Ross told Daryel *he* was able to twitch *his* nose.

(2) Syntactic constraints: in section 1, we have seen gender and/or number constraints. Now let's look at some other examples. The most obvious constraint is reflexivization. Consider:

3-7. Nadia says that Sue is knitting a sweater for *her*.

Her is Nadia or, in the right context, some other female, but cannot be Sue, as the syntax requires the reflexive herself to be used if Sue is the intended referent.

Another constraint prohibits a pronoun in a main clause referring to an NP in a subsequent subordinate clause:

3-8-1. Because Ross slept in, *he* was late for work.

3-8-2. Because *he* slept in, Ross was late for work.

3-8-3. Ross was late for work because *he* slept in.

3-8-4. *He* was late for work because Ross slept in.

In the first three sentences, he and Ross can be coreferential. In (3-8-4), however, he cannot be Ross because of the above constraint, and either he is someone in the wider context of the sentence or the text is ill-formed.

(3) Semantic defaults: The typical defaults are implicit verb causality, which can affect the antecedents assigned to nearby pronouns. For example, consider these texts:

3-9. Muriel won the money from Helen because *she* was a skillful player.

3-10. Ronald scolded Joe because *he* was annoying.

People tend to interpret she in (3-9) as Muriel, the first NP of the sentence, and he in (3-10) as Joe, the second NP. In general, with sentences of the form:

NP1 VERBed NP2 because { *he* | *she* }....

(where both NP1 and NP2 are of the same gender as the pronoun) there is a distinct tendency for people to construct and interpret the sentence such that the pronoun refers to NP1 in the case of some verbs, and NP2 in the case of some others. (some verbs are neutral) The strength of this tendency is the verb's causality.

We can see that if an NLU system had the implicit causality of each verb marked in its lexicon, this information could be used to help find the preferred antecedent in potentially ambiguous cases.

(4) Discourse structure and world knowledge: In the section 2, we introduced two important ideas in discourse: theme and focus. They themselves provide the structure for pronoun references, and, together with semantic network of general knowledge related to focus, provide an inferring mechanism to infer from general knowledge and some suppositions that a certain proposition is true. So the focus and the theme serve as a context mechanism in and of themselves. There are also default rules for context such as: the speaker who requests a meeting to be scheduled is likely to be a participant.

4. A New Pronoun Resolution System Based On Theorist Framework

From section 2, we know that modern approaches to pronoun resolution should be based on discourse structures, especially the theme and the focus. Also we gave a formal description of the nominal focus set of Fn and said we are most interested in case 3 a) that we need to choose one of the possible members of Fn as the antecedent of a pronoun reference. Each choice should fit the basic constraints ,and the best choice should fit more defaults as well. From section 3 , we see that there are different levels of constraints and defaults which can be used to help resolve the ambiguous references. All of these discussions imply that in order to find an appropriate antecedent of a pronoun in the focus set, we need make inferences based on constraints and defaults and decide the best antecedent among many choices. This leads us to present a pronoun resolution system which is based on Theorist Framework.

The following figure shows an overview of the pronoun resolution system. It consists of four parts: Database, Theory Formation/Preference, Meta-control, and Input.

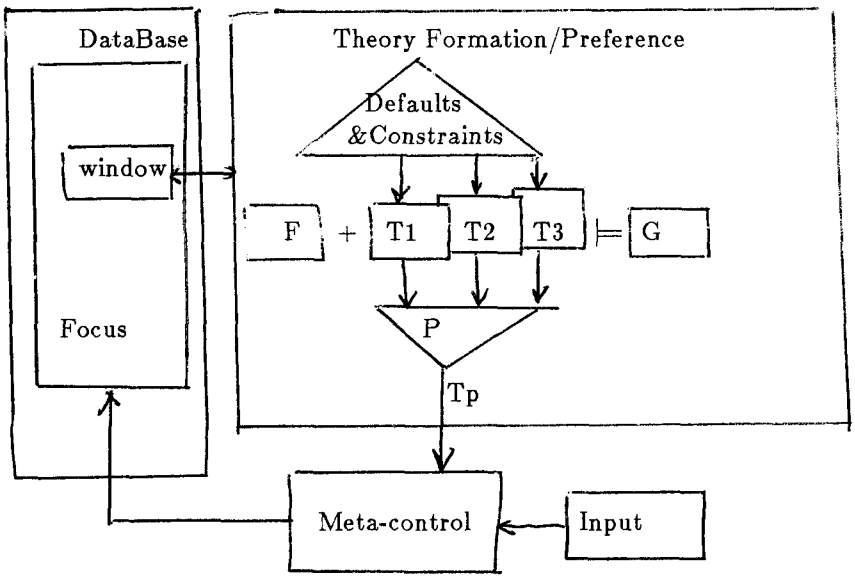


Figure 3. An overview of a new pronoun resolution system

The Input is a grammer recognizer. It performs the grammar checking over an input sentence and send it to the Meta-control. The Meta-control is the most important module in this system. Its first function is to decide the themes over input sentences and manage the focus set in the Database. Its second function is to control the pronoun resolution by following steps:

(1) It use discourse-level defaults to choose the members in the focus set as possible antecedents of the pronouns. These members in the focus set, together with the general knowledge related to them in the Database, form the window for the Theory Formation/Preference.

(2) If there is only one possible choice, which fits the basic constraints, it is chosen as the referent.

(3) If there are many choices, the module will make the best choice based on the results of the Theory Formation/Preference.

The Theory Formation/Preference is based on Theorist Framework([3],[4]). It uses the members and their related knowledge in the window of the focus set as facts F, the input sentence with pronouns as observation G. In addition, the constraints and defaults from the lexical to semantical levels form the hypotheses set Δ. The module performs default reasoning and generates a set of theories Ti, which is a set of instances of constraints and defaults. Then the module executes the procedure P for choosing a preferred theory Tp and send it to the Meta-control.

The Database contains the organized information about the focus, the discourse and real world knowledge. In our system, we prefer to use frames to represent the knowledge in the Database. A frame is a generalized property list where each generalized property is called a slot. It provides a structure within which new data are interpreted in terms of concepts acquired through previous experience. For example, a simple frame for the generic concept of chair might have slots for number of legs and style of back. A frame for a particular chair has the same slots--- they are inherited from the CHAIR frame--- but the contents of the slots are more fully specified([9]):

CHAIR Frame

Specialization-of: FURNITURE
Number-of-legs: an integer (DEFAULT=4)
Style-of-back: straight, cushioned, ...
Number-of-arms: 0, 1, or 2

JOHN'S-CHAIR Frame

Specialization-of: CHAIR
Number-of-legs: 4
Style-of-back: cushioned
Number-of-arms: 0

The theme of a discourse can be represented as a frame which contains in the discourse slot some information about what kinds of local themes (sub-theme) can be used to extend the discourse. A theme can have potentially many kinds of sub-themes, which are also represented as frames. For example, scheduling as a theme can have meetings, parties, lectures, etc as sub-themes. So from the discourse slot of the theme frame, we can know expectations of what the remaining sentences of the discourse are likely to discuss. The frame structure limits the type and number of concepts that can be discussed for a given theme. However, this is a productive limitation since it provides a limited number of ways to interpret subsequent sentences and can help the Meta-control to manage the focus set.

The Meta-control has two main functions. The first one is related to the determination of the contents of the focus sets. This is a difficult problem since we need another set of rules to deduce the themes of the discourse. The other function is about pronoun resolution. This actually includes two levels of referencing: intra-sentential and extra-sentential. Intra-sentential referring is needed as (4-1) and (4-2) below show:

4-1. *He* said *he* was going.

4-2. John washed *himself.*

Without intra-sentential referencing, the co-reference of himself to John and possible co-reference of the two he in (4-1) would be impossible. This implies that in resolving the reference we should first consider the intra-sentential referencing. For the extra-sentential referencing based on focus set, there are also

some choices of the members from the focus set which are more likely to be the antecedents of pronouns than the other choices. This information can be represented as discourse-level defaults in the Meta-control and will help make the best choice from the results of the Theory Formation/Preference. In general, the Meta-control is a very complicated module. What it should consist of and how it could be implemented would largely depend on the results of the discourse study.

In this paper, we are mostly concerned with the Theory Formation/Preference module, since it is based on the Theorist Framework and is relatively easy to handle.

5. Theory Formation And Theory Preference

The Theory Formation/Preference module is based on the Theorist Framework, which views reasoning as scientific theory formation (rather than as deduction). A typical theorist knowledge base consists of a collection of formulas classified as possible hypotheses (Δ), facts (F), and observations (G). (See the figure below, which is adopted from [7]).

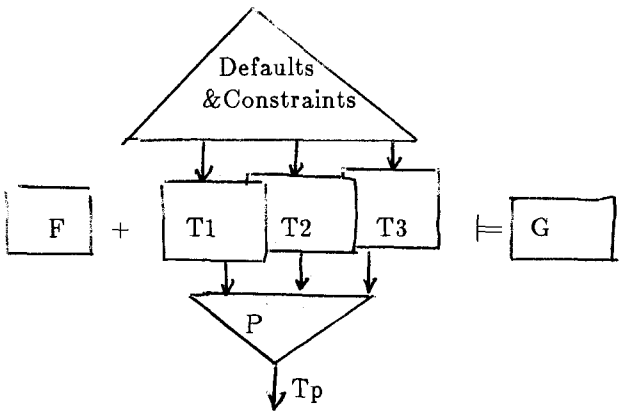


Figure 4. The Theory Formation/Preference module

The Theorist reasoning strategy attempts to accumulate consistent sets of facts and instances of hypotheses as explanations for which the observations are logical consequences. A theory is a subset of the possible hypotheses which are consistent, and imply the observations. More formally, if there is some subset D of Δ such that:

F u D $\models$ G and
F u D is consistent.

then we say G is explanable and D is a theory that explains G. For example, suppose we have:

Δ = { birdsfly(X): flies(X) <- bird(X); }
F = { n(flies(X)) <- emu(X);
bird(tweety); }
G = { flies(tweety); }

then flies(tweety) can be explained since there is consist theory birdsfly(tweety) which can explain flies(tweety). But when we add the fact emu(tweety), that theory is no longer consistent, as its negation can be proven.

In our system, Δ is a collection of constraints and defaults from the lexical level to the semantic level (See section 3). G is the input sentence with pronouns and F is some members and their related knowledge in the window of the focus set. For a given pronoun in the input sentence, the Theory Formation/Preference will make default reasoning on ^ and F, and try to find the possible theories which can explain G as referent of the pronoun. The problem here is that in many cases, we will obtain more than one theory, which correspond to the ambiguous references in the discourse. Thus in order to disambiguate the references, we need a procedure P for choosing a preferred theory Tp. This makes the Theory Formation/Preference take the same form as that in the paper [7], since they both make theory

preference based on theory framework. The theory preference is, in general, a difficult problem, but fortunately in our application the things are not that difficult. We find there exists a natural measure, the number of rules in a theory, which can be used in theory preference. Intuitively, if the substitution of some concept to a pronoun fits more constraints and defaults, we will prefer this concept as the referent of the pronoun. In other words, the more the number of rules a theory has, the more it is preferred. For our human hearer, this seems also the case, since we always use more constraints to help resolve the ambiguities. In the example (5-1) below:

5-1. Nadia says that Sue is knitting a sweater for *her*.

her is ambiguous for referencing to Nadia and Sue, though both theories which explain Nadia and Sue will fit the gender and number constraints. However if we add the constraint of reflexivization, Nadia as the referent to her is preferred because the theory which explains Nadia has one more rule than the theory which explain Sue.

In general, we find our theory preference measure will be helpful in resolving three kinds of reference problems.

(1) Exceptions to the constraints:

In the section 1 and 3, we said that pronouns are usually marked for gender and/or number. However these strong constraints have awkward exceptions. For example,

5-2. How is Swiss? *She* is a beautiful country.

5-3. The author thanks the reader for *their* kind indulgence.

In the above examples, a feminine pronoun "she" can have a neutral antecedent "Swiss", and a plural pronoun their a singular antecedent the reader. If we only use gender and number constraints, the reference to Swiss of she and the reader of their would not be explainable. But if more defaults are added, these two exceptions could be handled. In the example (5-2), we know that the theory which doesn't explain Swiss for she is based on the gender constraint. However the theory which explain Swiss for she satisfies the number constraint. Also from the second sentence, we know that she is a country. If we add a semantic default that a pronoun can refer to some concept with a same property, then the theory which explain Swiss for she will be preferred because Swiss is also a country and will fit the semantic default. In the example (5-3), the reader and their don't satisfy the number constraint, but they fit the gender constraint. If the default about thanks is added, the theory which explains the reader for their is preferred, since thanks is usually used in the structure "thanks one for one's something".

(2) Some ambiguous references

In section 3, we mentioned that some sentences are genuinely ambiguous if they don't appear in an appropriate context. For example:

5-4. Ross told Daryel *he* had passed the exam.

5-5. Ross loves *his* wife and Daryel loves *his* wife too.

In (5-4), he would be either Ross or Daryel and in (5-5), Daryel would love either his own wife or Ross's. However, some which are theoretically ambiguous could be resolved with the help of some defaults. In the following example,

5-6. Daryel told Ross *he(1)* was the ugliest person *he(2)* knew of.

each occurrence of he could mean either Daryel or Ross, giving a total of four readings for the sentence. Yet if we add a default that "if X told Y something, X usually told something X knew of", then the he(2) should refer to Daryel. Now the theories for (5-6) would be that Daryel is insulting Ross (he(1) = Ross, he(2) = Daryel) and that Daryel is self-critical (he(1) = he(2) = Daryel). Since insulting behaviour is more common than openly self-critical behaviour with respect to personal appearance, the first theory will be preferred, that is, (he(1) = Ross, he(2) = Daryel).

(3) Ill-formed sentences

In the section 3, we also gave an example of ill-formed sentence which doesn't fit the gender constraint.

5-7. Sue find *himself* pregnant.

Although the theory that explain Sue for himself satisfies the number constraint, the theory which doesn't explain Sue for himself will fit more constraints and defaults, such as the gender constraint and the default that "if X is pregnant, X must be female". Also himself is used in this sentence, its referent must be somewhere in the sentence, but we couldn't find it. Thus we decide this is an ill-formed sentence.

The theory Formation/Preference is under control of the Meta-control. Its main function is to produce a preferred theory based on the constraints and defaults, and the facts in the window of the focus set, observations from the Input. The Meta-control has the measures for the results from the Theory Formation/Preference will give another set of members in the focus set as a window and call the Theory Formation/Preference again. Otherwise the Meta-control will begin to process another input sentence.

6. Summaries And Suggestions

From the above discussions, we can see that the modern approaches to pronoun resolution should be based on the structure of a discourse, especially the theme and the focus, since they basically model the consciousness of the hearer as a repository for antecedents. Besides this, we also know that there are a lot of constraints and defaults from the lexical level to discourse level, which could be used to make inferences to resolve pronoun references. Based on these ideas, we present a new pronoun resolution system which consists of two levels of control: Meta-control and Theory Formation/Preference. The Meta-control can decide the change of the themes in the discourse and manage the focus set in the Database. It also uses discourse-level of defaults to control the pronoun resolution over the Theory Formation/Preference. The Theory Formation/Preference is based on the Theorist Framework, which takes the constraints and defaults from the lexical level to semantic level as the hypotheses set and generates a group of consistent theories by using default reasoning. These theories need to be measured to select the preferred one and this , in general, is a difficult task since it need a lot of domain-specific heuristics. Fortunately, in our system, we find the number of rules in a consistent theory could be used as a natural and productive measures, and we show this by giving a number of examples, especially for three cases: exceptions to constraints, some ambiguous references and ill-formed sentences.

Meanwhile we should admit that our consideration for this system is only immature and far from completion. There is much work to be done about it. First of all, we need consider more about the implementation of the system and test wide-range discourses to see to what extent the system could resolve pronoun references. Then we need extend the system to resolve more general references of anaphors, or even further, the definite noun phrases. Last if we hope this system more useful , we might also need to consider the relation of reference resolution to the discourse understanding, which includes many hard problems in natural language understanding, such as Speech Act Interpretation, Deep-inferring, Intentions, etc.

References

[1] Candace Bullwinkle, "Levels of Complexity in Discourse for Anaphor Disambiguation and Speech Act Interpretation", Proceedings to the Fifth International Joint Conference in Artificial Intelligence, August 1977.

[2] Candace L. Sidner, "The Use of Focus as a Tool for Disambiguation of Definite Noun Phrases", TINLAP-2, 1978.

[3] David L. Poole, "Theorist: a logical reasoning system for defaults and diagnosis", Depart. of Computer Sci., Uni. of Waterloo, Technical Report CS-86-06, February 1986.

[4] David L. Poole, "Default Reasoning and Diagnosis as Theory Formation", Depart. of Computer Sci., Uni. of Waterloo, Technical Report CS-86-08, March 1986.

[5] Graeme Hirst, "Anaphora in Natural Language Understanding: A Survey", Springer-Verlag, 1981.

[6] Raymond C. Perrault, "An Application of Default Logic to Speech Act Theory", AI Center and CSLI, SRI International, 1986.

[7] Scott D. Goodwin and Randy G. Goebel, "Theory Preference based on Preference", Depart. of Computer Sci., Uni. of Waterloo, 1986.

[8] Yorick Wilks, "A Preferential, Pattern-seeking, Semantics for Natural Language inference", AI. Vol. 6, 1975, pp53-74.

[9] Avron Barr and Edward A. Feigenbaum (ed.), "The Handbook of Artificial Intelligence: Vol. 1", William Kaufmann, Inc.