"Inductive Inference of Theories from Facts"の日本語訳

北上　始
ICOT 2nd Research Laboratory

［概要］

本報告は、１９８１年に、Ehud Y. Shapiro 博士がエール大学でまとめた研究報告書（事実による理論の帰納推論）に対する、ほぼ全訳（日本語訳）である。

彼は、"事実による理論の帰納推論"を、モデル推論と呼んでおり、その理論的な枠ぐみの定義からはじめ、帰納推論の一般的で増殖的なアルゴリズムを、与えている。

また、彼は、ロジックプログラムの虫取りという立場から、そのアルゴリズムをとらえ、そのアルゴリズムが、主に、(i) 矛盾探索(Contradiction Backtracing) アルゴリズムと、(ii) 仮説生成スキーマにより実現されると、みなすことができると、述べている。(i) は、虫の所在位置を見つけるアルゴリズムであり、(ii) は、虫の修正方法についての枠ぐみを与えるスキーマである。彼は、このスキーマを、精密化演算子(Refinement Operator) と呼び、そのいくつかの有益な性質を、述べている。

最後に、このモデル推論の実現手段として、ロジック的アプローチが成功した理由を、述べている。

なお、この日本語訳は、Preliminary Version であり、予告なしに修正される。

Inductive Inference of Theories From Facts

February, 1981
Research Report 192
Ehud Y. Shapiro

Department of Computer Science
Yale University

日本語訳 Preliminary Version

初版 '83.1/20
第2版 '83.5/09
第3版 '83.7/29
第4版 '83.9/07
第5版 '84.1/11
第6版 '84.3/30

ICOT 2nd Research Laboratory
Hajime Kitakami (北上 始)

目　次

事実による理論の帰納推論

概要

本論文は、モデル推論の問題とアルゴリズムについて述べられている。モデル推論問題は、科学者が直面している問題の抽象化である。科学者は、いくつかの限定されたわく組の下で、いくつかの領域を研究し、実験を行う。そして、その結果を説明できる理論を見つけようとする。これを抽象化すると、探索領域は、1階述語言語Lに対するいくつかの未知モデルMの領域であり、実験は、Mにおいて、Lの構文に関する正当性をたしかめることである。我々の目的は、例文が全て真なる仮説の集合を見つける事にある。

本論文の主な成果は、モデル推論問題を解くための、一般的で増殖的なアルゴリズムである。モデル推論問題は、推測と反ばくからなるPopperの方式［Popper 59, 68］に基づく。このアルゴリズムは、ある種別の複雑なクラスのモデルで、任意のモデルをある極限［Gold 67］で同一視するように示される。また、本アルゴリズムは、その中で最も有効で、かつ、いくつかの具体的領域で首尾よくインプリメントされるのに十分に柔軟なものである。

本モデル推論アルゴリズムは、二つの調整可能なパラメータをもつ。一つは、仮説の構造をいかに複雑にするかであり、他の一つは、仮説からどの程度複雑な導出をさせるかである。両者は、同時に、このアルゴリズムの制限下で、帰納的に推論されるあるクラスのモデルを決定している。一方では、両パラメータは、このモデル推論アルゴリズムが、任意に決められた帰納関数で構成されたモデルの極限で同一視されるように定められる。他方ではそれらは、近似的にインプリメントされたアルゴリズムが、実用時間内で、少ない事実から具体的なモデルの公理化を推論するように定められる。

このモデル推論システムは、ホーン形式の理論を推論するように限定されたモデル推論アルゴリズムを、基礎にしている。本システムは、プログラミング言語Prolog［Pereiea et al. 78］で、インプリメントされた。例として、算術式の領域で、本システムは、36個の事実からFigure１－１の下に書いた公理の集合を、27 sec（ＣＰＵ）で導いた。本システムは、終端をもつ稠密な半順序に対して、ある公理化を発見した。それは、append, reverseそして、夏の課題論文で述べていた［Summers 76］例のほとんどに対し首尾よく、ロジックプログラムを合成した［Kowalski 79a］。

また、boolean formulas, binary tree, inclusion, binary tree isomorphism その他についても満足のいくロジックプログラムを合成した。

この一般的アルゴリズムの一部分として、矛盾を探索するアルゴリズムが、発見された。このアルゴリズムは、ある矛盾が推測と事実に生じるときはいつでも

本アルゴリズムは、この矛盾の原因を後もどりで見つけられる。例えば、この矛盾の原因は、ある偽の仮説である。また、本アルゴリズムは、ある反例を作成する事により、仮説の虚偽性を示すことができる。その様なアルゴリズムの存在は、科学的理論［Harding 76］の反ばくに関する哲学的議論に見られる。また、ホーン理論の制限は、ロジックプログラムの虫取りに対し、実用性を促すかもしれない。

１．はじめに

本論文は、以下の形の問題を扱う。

我々は、あるたしかな領域から実際の知識を得ることができるとする。ただし、この領域は、いくつかの未知のルールにより支配されており、これらのルールを表現するのに適切な言語が与えられていると仮定する。

我々は、いかにして、この実際的な知識からこれらのルール又は理論を見つけるのだろうか？

これは、ある決められた概念の中で、いくつかの領域を研究している科学者が直面している問題である。即ち、彼らは、実験を行ったり、領域内で真なる定理を見つけようとしたり、それらの結果を説明しようとしたりしている。モデル推論問題は、この立場の抽象化である。本節では、例でモデル推論問題を説明する。

これらの例は、モデル推論問題が解けるアルゴリズムによる可能な応用を提案している。この研究の関連性を、科学的発見の問題で議論し、本論文の構成を与える。

１．１　いくつかのモデル推論問題

Figure 1-1にモデル推論問題の例を示す。

探索領域は、整数であり、与えられた１階述領域の言語は、定数０、後続関数Ｘ'、そして以下の３つの述語を含む。

Ｘ≦Ｙ……ＸはＹより小か等しい

plus（Ｘ，Ｙ，Ｚ）……Ｘ＋Ｙ＝Ｚ

times（Ｘ，Ｙ，Ｚ）……Ｘ×Ｙ＝Ｚ

これらの関係が、具体的数値で成立するかどうか確めてみよう。即ち、Ｍで真なる０≦０'''，plus（０'，０'，０''），times（０''，０''，０'''）としてのground atom（variable-free）を確めることができる。この立場では、モデル推論問題は、構文の有限集合を見つけることにある。この構文は、正しい算術式であり、すべて真のground atom を導いている。Figure1-1 は、その構文の一組を示している。逆の矢印“←”を“～により証明される”という記号とする。構文Ｐ←Ｑ＆Ｒは“ＰはＱとＲの結合により算術される”と読む。

Figure 1-1：算術式の推論

領　域：整数

言　語：0　　－　ゼロ。
　　X'　　－　Xの後続要素。
　　X≦Y　　－　Xは、Y以下である。
　　plus（X，Y，Z）　－　X＋Y＝Z。
　　times（X，Y，Z）　－　X×Y＝Z。
事実の例：0≦0'は、真である。
　　plus（0，0'，0）は、偽である。
　　times（0''，0''，0''''）は、真である。

理　論：X≦X
　　X≦Y' ←X≦Y

　　plus（X，0，X）
　　plus（X，Y'，Z'）←plus（X，Y，Z）

　　times（X，0，0）
　　times（X，Y'，Z'）←times（X，Y，W）&
　　　　plus（W，X，Z）

このモデル推論問題を解くために、関数と述語の性質をすべて見つけ出す必要がないことに注意しよう。特に、上記、理論は、加法の連結性と順序集合の推移性を含んでいない。もし、Tが、Mで真なるLのground atom を全ての導出するMで真なる構文であれば、TはMに関し atomic-完全公理化 (atomic-complete axiomatization) であると呼ばれる。Figure1-1 の構文の集合は、算術式に関しatomic -完全公理化である。それは、３６の事実から２７CPU secondで、モデル推論システムにより推論された（付録Ｉ参照）。

我々は、１階述語の言語Lで２つの形の構文を区別する：それらは、実験結果の記述に関連する観測文、及びそれらの結果の説明として与えられる仮説である。科学者の探索領域は、Lに関するいくつかの未知モデルMの領域である。この領域で行なわれている実験は、Mで真なる観測文を調べることにある。事実は、その様な実験の結果の陳述である。モデル推論問題は、以下の科学的発見の問題に関連している。

いくつかの未知モデルMの実験を行う事ができる能力を与えると、Mで真なる仮説の有限集合が見つけられる。ただし、その仮説は、すべての観測文を真にする。

仮説に使われる言語を固定するという過程は、科学的発見の主要問題の一つを回避することになる。この問題は、新しい概念スキーマの発明である。Kuhnianの用語［Kuhn 70 ］では、モデル推論問題は、新しい概念スキーマを見つけるよりは、“normal science”のパズル解決の方式に似ている。新しい概念スキーマは、“凡例置き変え”過程という特性がある。

他のモデル推論問題が、Figure 1-2に示されている。この例では、探索領域は、２進ストリングの集合である。また、この１階述語の言語は、単項述語in（Ｘ）、２つの後継関数０（Ｘ）、１（Ｘ）及び、null string を示す定数Λを、含む。この言語では、項（１（０（０（１（Λ））））を、ストリング１００１で示し、１（０（Ｘ））の前置オペレータをストリングで示す。かっこは、通常除かれる。我々は、具体的なストリングが、いくつかの未知の集合の中に存在するかどうかを確められると仮定する。また、我々のゴールは、この集合中のすべてのストリングに対し、in（Ｓ）を証明し、他の任意の２進ストリングに対し、～in（Ｓ）を証明する。正しい構文の有限集合を見つける事にある。

Figure1-2 ：ある形式的言語の推論

領　　域：２進ストリング

言　　語：Λ　　－　空ストリング。
　　０（Ｘ）　－　Ｘに０を結合する。
　　１（Ｘ）　－　Ｘに１を結合する。
　　in（Ｘ）　－　Ｘは、偶数の０と１の偶数の１をもつ。

事実の例：in（００１１）は、真である。
　　in（０１１）は、偽である。
　　in（０１０１１）は、偽である。
　　in（０１０１００）は、真である。

理　　論：in（Λ）
　　in（００Ｘ）←in（Ｘ）
　　～in（０Ｘ）←in（Ｘ）
　　in（１１Ｘ）←in（Ｘ）
　　～in（０１Ｘ）←in（Ｘ）
　　～in（１Ｘ）←in（Ｘ）
　　～in（１１Ｘ）←in（０Ｘ）＆　～in（Ｘ）
　　～in（０１Ｘ）←in（０Ｘ）＆　～in（Ｘ）
　　～in（１Ｘ）←in（０Ｘ）＆　～in（Ｘ）
　　～in（１０Ｘ）←in（１Ｘ）＆　～in（０Ｘ）＆　～in（Ｘ）
　　～in（００Ｘ）←in（１Ｘ）＆　～in（０Ｘ）＆　～in（Ｘ）
　　in（００Ｘ）←～in（１Ｘ）＆　～in（０Ｘ）＆　～in（Ｘ）
　　～in（００Ｘ）←～in（１Ｘ）＆　～in（０Ｘ）＆　～in（Ｘ）
　　in（１０Ｘ）←～in（１Ｘ）＆　～in（０Ｘ）＆　～in（Ｘ）

もし、未知の集合が十分に簡単であれば（もっと正確にいうと、それは規則的である…[Angluin & Hoover 80]）、この言語の中に、有限集合の公理が存在する。もし、TがM上の構文で真であれ* ば、TはMのground -完全公理化（ground-complete axiomatization）であると呼ばれている。Figure1-2 は、偶数個の０と１をもつストリングの集合に対するground -完全公理化を示す。この特殊な公理は、Angluin とHooverのアルゴリズムによるこの集合に対し、４状態のアクセプターから作成された。モデル推論システムの古い判は、６４個の事実から３分CPU でこの集合のground -完全公理化を推論した。本システムにより発見された公理は、各条件に１つのアトムを含む。その条件は、上記のいくつかの公理の２と３に対立する。本アルゴリズムのふるまいは、任意の規則集合が、ground -完全公理化をもつことを提案している。この結果は、どこかよそで [Angluin & shapiro 81] 記述されるだろう。

同じ規則集合のatomic -完全公理化を以下に示す。それをFigure 1-2の公理化と比較せよ。Figure 1-2の公理は、１つのモデルに関し、atomic -完全公理化と、ground -完全公理化の区別を明確にできる。モデル推論システムは、２８ CPUでこの公理に追いついた。同時に、３５個の事実を読んだ。

in（Λ）
in（００X）←in（X）
in（１１X）←in（X）
in（０１０X）←in（１X）
in（０１１X）←in（０X）
in（１００X）←in（１X）
in（１０１X）←in（０X）

規則性をもった帰納的推論問題は、GoldとAngluin [Gold 78,Angluin 78]らによって研究された。それは、モデル推論アルゴリズムとシステムを開発する為のある動機づけとテストケースとして役立っている。

我々のフレームワークに自然に一致するモデル推定問題の第３番目は、自動プログラミングの領域が取り上げられる。また、それは、いくつかの例によるプログラム合成を意味する [Green et al. 74,Summers 76, Summers 77, Bierman 78]。その仕事は、入出力のふるまいから、帰納的にプログラムを推論することにある。この仕事は、モデル推論問題として言いかえることができる。また、この場合、推論されるプログラムは、ロジックプログラムである。

---

* 任意のground atom P ∈ L に対し、PがMで真であり、それ以外で～Pであるならば、TはPを証明する。

Figure1-3 ロジックプログラムの推論

領　域：リスト

言　語：[]　　－ nil
[X | Y]　　－ XとYの“cons”
append (X, Y, Z)　－ XをYに加えるとZになる。
reverse (X, Y)　－ Xは、Yの反転である。

事実の例：append ([], [a], [a]) は、真である。
append ([a, b], [c, d, e], []) は、偽である。
append ([a, b], [c, d, e], [a, b, c, d, e])
は、真である。
reverse ([a], [b, a]) は、偽である。
reverse ([a, b, c], [c, b, a]) は、真である。

理　論 append ([], X, X)
append ([A | X], Y, [A | Z]) ←append (X, Y, Z)

reverse ([], [])
reverse ([A | X], Y) ←reverse (X, Z) &
append (Z, [A], Y)

ロジックプログラムは、ホーン節の集まりであるが、その形式は、P←Q_1 & Q_2 &……&Q_n である。PとQは、アトムである。この構文は、“Pは、Qの結合により証明される”と読み、手続き的には、“ゴールPを満すためには、ゴールQ_1，Q_2，……，Q_n を満すことである”と解釈される。ホーン節の集まりは、手続き的解釈を使ったプログラムとして実行される。通常、そのようなプログラムには、効率を考慮した、制御情報が記述されるべきである。ロジックプログラムの紹介として、[Kowalski 79a] を参照せよ。
また、ロジックプログラムの操作的でかつ表示的意味についての議論に対しては[Van Emden & Kowalski 76]を参照せよ。アルゴリズムのロジックと制御の間の区別の苦心については、[Kowalski 76b] を、参照せよ。
Prologマニュアルとしては、プログラム言語に基づくロジックの具体的インプリメントの詳細[Pereira et al. 78]がある。

我々の設定では、もし、仮説に使われる言語が、ホーン節に、制限されれば、atomic -完全公理化は、Lの述語を計算すめためのロジックプログラムになる。Figure1-3 は、そのようなプログラムの例である。この例では、言語Lは、2変数関数［X｜Y］、定数［］、そして、二つの述語append（X，Y，Z）とreverse （X，Y）を含んでいる。この言語に対するモデルMは、次のように定義される：Mの要素は、［X｜Y］と［］から構成されるリストである；そのアトムappend（X，Y，Z）は、Mで真である。この場合、リストZはリストXにYを付加したリストに等しい。
例えば、append（［a，b，c］，［d，e］，［a，b，c，d，e］）：
アトムreverse(X，Y）は、Mで真である。Yは、Xを反転したリストに等しい。
Figure1-3 のホーン節は、このように定義されたatomic -完全公理化である。ふり返ると、これらの節は、appendとreverse に対してのPrologプログラムである。例えば、リスト［a，b］に［c，d，e］を付加した結果を計算する為に、goal←append（［a，b］，［c，d，e］，X）は、上記プログラムをロードされたPrologインプリンタに与えられるが、X＝［a，b，c，d，e］が、返される。モデル推論システムは、３６の事実から１１ CPUでappendに対するロジックプログラムを合成し、１３個の事実から６CPU でreverse に似たProgram を合成した。ロジックプログラムのさらに進んだ例は、本論文の至る所で与えられている。また、Appendix I に、例による、いくつかの合成の説明が見られる。

Concept-learning tasksの立場から、ＡＩのいくつかの研究は、モデル推論の問題として言替えられる。例えば、archesとnon-archesの記述から、arch［Winston75 ］の概念を学習する問題は、以下のように言い換えられる：探索領域は、複数ブロックをつみ重ねるすべてのオブジェクトである。言語は、block(X）、column（X）、on（X，Y）、arch（X，Y，Z）のような述語を含む。

帰納推論問題は、構文の有限集合を見つける事にある。その構文は、合成オブジェクトがarchesである事を決めることができる。archesは、archesとnon-archesの例により与えられる。そのような構文の集合は、Kowalski [Kowalski 79a] により提案された：

```
arch(X,Y,Z) ←block(X)& column(Y)& Column(Z)&on(X,Y)&on (X,Z)

column(X) ←block(X)
column(stack(X,Y))←block(X)&column(Y)&on(X,Y)

on(X,stack(Y,Z))←on(X,Y)
```

この構文の集合と、次式から、

```
block(a)      on(a,b)
block(b)      on(b,c)
block(c)      on(a,d)
block(d)      on(d,e)
block(e)      on(e,f)
block(f)
```

arch (a,stack(b,c),stack(d,stack(e,f))) を証明できる。一般に、あるシーンがarchの概念のあるインスタンスになっているがどうかは、各公理から決定される。また、そのような公理を要求することにより、この概念を学べることは、妥当であるように見える。

１．２　科学的発見の問題について

モデル推論問題に対する、任意の興味ある解は、ある予備的問題と折合いが付かねばならないが、それは、帰納の問題である。この問題は、哲学者David Humeにまでさかのぼる。帰納の問題を示すと、有限の量をもつ実際のデータは、もちろん無限の結果をもつ理論を決して確立することができないといえる。この問題の魅力的な解は、Karl.R.Popper （哲学者）によって提案された［Popper 59］。Popperは、実際のデータによる一般的科学理論が、不確定であることを述べたHumeの考えを認め、科学理論の推測的立場が認識される事を提案した。科学の有効性は、理論がまちがいであれば、実際にそれは拒絶されるだろうという事実に基づいている。一般に、その理論が正しい限り、もちろん改良されない。

帰納の問題に対するPopperの方式は、科学行動に対し、いくつかの手続き的な忠告を生ずる。Popperは、実験行動が、提案された理論を拒否する力をもつ新しいデータを集める方向にある間、理論的科学行動が、簡単で容易に反ばく的な理論を作成する方向に動くべきであると提案した。ただし、この理論は、既知のデータを、説明している。Popperは、理論と実験の間、すなわち、推測と反ばくの間の相互作用は、真実へのある種の収束現象を導びいてるのかもしれないと主張した。　真実らしさ［Popper 68］の概念は、競争的な科学理論がもつ“もっともらしさ”を測定したり、比較したりする事により創案された。非公式には、T_1がT_2と同じくらい多くの真の観測文を導びき、T_2よりも多くの偽となる観測文を導びかないとき、仮説T_1の集合の真実らしさは、T_2の真実らしさより大きいか等しいかである。それらの真実らしさを測定しする理論的数値をシステム的に割り付ける事ができる。便宜上、この値は、０と１の間の範囲とする。この様な測定下では、すべて真の観測文を導びく真の理論は。真実らしさ１をもち、矛盾しかつ反復的な理論は、真実らしさ０をもつ。科学的発見の過程は、その理論の真実らしさを増加する永続的試みであるとして、見ることができる。

真実らしさの概念は、諸理論を競争比較するのに使われる。しかしながら、競争的な科学的発見の方法論又は帰納的推論アルゴリズムのいくつかを評価・比較するために、その着目点が単一理論よりもむしろ科学的な発見プロセス又はアルゴリズムに存在するように思われる。帰納的推論アルゴリズムの能力を評価、特徴づける方式は、Gold［Gold 65］によって提案され、極限における同一視と呼ばれている。この概念の理論的根拠は、Popperのそれと良く似ている：一つのモデルについての有限の事実の数は、一般的に、すべての可能なモデルの間では、ユニークに決められない、また帰納的推論アルゴリズムおよび科学者は、いつも有限の事実を使って推測を行うので、それらのどちらもあやまちをおかさざるを得ない。

ある意味では、これはPopperの命題に関する難問である。帰納的推論アルゴリズムで期待できる事は、次のようなことである。そのモデルについて有限の事実をたしかめ、有限のまちがった推測を行った後、そのようなアルゴリズムは、正しく、真なる仮説の集合を推測するだろう。そして、その仮説は、全て真なる観測構文を導くことになるだろう。そのような場合、我々は、このアルゴリズムが極限でそのモデルに同一視されると言う。その様な仮説の集合は、真実らしさが1であるが帰納的推論アルゴリズムは、一般に、その様な仮説の集合を実際に見つけたかどうかを決められないことに注意しよう。極限における同一視の概念は、帰納推論での帰納理論と複雑さの理論の研究で実りが多いことを証明した［Gold 67,Blum & Blum 75,Case & Smith 81］。

1．3　本論文の構成

第2節では、モデル推論とアルゴリズムの概念をもっと厳密に定義する。このとき、［Blum & Blum 75］の方式に従い、いくつかの複雑なモデル推論問題を考察する。第3節では、いくつかの帰納関数hに対するいくつかのh-easyモデルのクラスを定義する。またあるhについて、極限で、任意のh-easyモデルを同一視できる枚挙型アルゴリズムを述べる。帰納的推論アルゴリズムが簡単で多くの要求を満足、即ち、その現在の推測では、いつも現在知られている事実を説明できるならば、そのようなアルゴリズムは、極限でh-easyモデルとだけに同一視できる。これらの結論は、任意に多くの帰納推論アルゴリズムの能力に上限を定めている。また、この結論は、この上限が、現実的である事を示している。

この一般的で複雑な理論的問題解決の方式は、任意の計算可能なモデルに適用できるがロジックの特別な性質を使っていない。この解析による動機づけを行うために、あるロジック指向のアルゴリズムと概念を開発している。このアルゴリズムと概念は、ロジックの意味と構文上の性質の比較的有利な点を取り出した比較的効率的で増殖性のある推論アルゴリズムの構成を提供できる。この増殖的なモデル推論アルゴリズムを開発するために、以下の2つの問題に注目する。

1．推測が非常に強くなったときに、いかにその推測を弱めるか？
2．推測が非常に弱くなったときに、いかにその推測を強めるか？

仮説がM上で偽なる観測文を証明するならば、モデルMに関して非常に強いと言う。仮説がM上で真なる観測文を証明することがないならば、非常に弱いと言う。

矛盾探索アルゴリズム（The Contradiction Backtracing Algorithm ）は、最初の問題に答えることを試みている。これは、第４節で述べる。このモデルで、ground atom が真実であることを判断する有限回の実験を行うことにより、このアルゴリズムは、推測と事実の間の矛盾の根源を逆にたどり、偽の仮説を選択することができる。

このアルゴリズムにより行なわれる種々の判断は、決定的な実験としての科学哲学で知られている。それらの重要性は、多くの方法論により認識できるが、偽の仮説の選択を保証する順序だてられたアルゴリズム的方式は、新規性をもつ。そのようなアルゴリズムの存在は、明らかにDuhem ［Duhem 54］の主張と矛盾する。彼の主張は。その可能性を、簡単に否定している。この主張は、科学的理論の非反ばく性を扱っているDuhem-Quine の命題［Harding 76］として、あとで何が知られるようになったかという事実から述べられた。矛盾探索アルゴリズムとそれを混合する一般的な帰納推論アルゴリズムの存在は、反ばく性の論争に関する哲学的論議を一新するかもしれない。矛盾探索アルゴリズムは、帰納推論の領域を越えた応用をもつ；我々は、ロジックプログラムの虫取りとしてその応用を提案している。

一度、推測の中で偽なる仮説を発見すると、とられるべく動作は、この仮説をそこからとりのぞくことである。しかしながら、これにより得られる推測は、すでに真である観測文を証明するのには弱すぎるかも知れない。この様な状況においては、第２の問合せが発生する。それを強くするために、その推測に、いかに新しい仮説を付け加えるかということを提案している。

精密化演算子(refinement operator) は、調整可能なこの推論アルゴリズムのパラメータである。さらに一般的な精密化演算子は、たとえ効率が小さいアルゴリズムになっても、より強力になる。あるクラスの仮説に対し、精密化演算子の完全性が定義され、種々のクラスの仮説を完全にするいくつかの精密化演算子が述べられる。非常に一般的な精密化演算子が定義され、任意の１階述語の言語に対して完全であることが示される。

これらの二つの問題の解決に基づいて、一般的で増殖的なモデル推論アルゴリズムが、第６節で開発されている。また、任意のh-easyモデルを、ある極限で同一視することも証明されている。また、いくつかのインプリメンテーション上の論点が論議される。本論文に述べられている陳述は、なぜこの要素を研究するのかを示している。また、付録Ｉは、そのアルゴリズムをインプリメントしたモデル推論システムの効率が述べられている。本システムは、次の論文でちっと十分に論議されるだろう。

２．問題の定義

本節では、もっと正確にモデル推論問題を述べ、モデル推論アルゴリズムの概念を定義する。ここでは、観測と仮説言語についての許容的な要請を紹介するが、この要請は、理論が事実によって反ばくされるべきであるというPopperの要請を現わしている。また、理論項の概念を含むモデル推論問題への拡張を議論する。

２．１　モデル推論問題とアルゴリズム

与えられる１階述語の言語Ｌは、有限に多くの述語と関数記号をもつ([Robinson 65]で定義されたような）節形式である。Ｌの構文は、次の形をしている。

$\{P_1, P_2, \cdots\cdots, P_j\} \leftarrow \{Q_1, Q_2, \cdots\cdots, Q_k\}$　　$j,k \geq 0$

ここで、P'sとQ'sは、アトムである。この概念は、P'sが正のリテラルであり、否定Q'sが節の負のリテラルであるという標準節形式の概念と、等価である。この構文は、Q'sの結合がPの選言を導出すると解釈される。Q'sは構文の条件と呼ばれ、P'sは、その結論と呼ばれる。空の条件は、少なくとも結論の中の１アトムが真であることを示す。空の結論は、条件の中の全てのアトムが真であることを示す。□は、空の構文であり、Ｌのモデルで偽を意味する。１アトムだけをもつ構文は、単一構文と呼ばれる。単一構文｛Ｐ｝←とアトムＰを区別しない。また変数の名前をつけるえると等しくなる構文も区別しない。標準的な節の概念では、構文に存在するすべての変数は、黙目的に全称的な量が定められている事に注意せよ。

Ｌの構文において、２つの部分集合を区別する：それは、観測文L_0と仮説文L_hである。$\square \in L_0 \subset L_h \subset L_1$であり、この両集合は、実質的に識別可能である。観測文は、L_0の構文であり、仮説は、L_hの構文である。探索領域は、Ｌの中のあるモデルＭである。また、観測文αが与えられると、もしαがＭで真ならば“true”、それ以外は“false ”を返すというある仕かけがあるとする。その仕かけは、ＭにとってのOracle（神託）とよばれる。そのOracleに入力を与え、その答を読む操作は、Ｍの実験と呼ばれる。領域Ｍについての事実は、形式$<\alpha, V>$の組である。この形式は、$\alpha \in L_0$を、観測文とし、
$V \in \{true, false\}$をＭにおけるαの真偽とする実験の結果である。モデルＭ上で真なる観測文の集合は、L_0^mで示される。ある完全な誘導手続きを仮定し、ＰがＴから導出される｛されない｝事を示すためにＴ⊢Ｐ｛Ｔ⊬Ｐ｝を使う。構文Ｓの集合からＴ⊢Ｓであるためには、すべてのＰ∈Ｓに対しＴ⊢Ｐでなければならない。それ以外は、Ｔ⊬Ｓである。

定義２．１：　L_0 と L_h をＬの部分集合とし、ＭをＬのモデルとする。ただし、$\square \in L_0 \subset L_h \subset L$ である。構文 $T \subset L_h$ の集合が、Ｍで L_0 -完全公理化であるための必要十分条件は、ＴがＭで真であり、$T \vdash L_0$ となる事である。

ここで、さらに厳密なモデル推論アルゴリズムの問題を定義できる。

１階述語の言語Ｌと、前述で定義された２つの部分集合（観測言語 L_0 と仮説 L_h ）を与えるとし、Ｌに関して、ある未知モデルＭに対するOracle（神託）が与えられているとする。モデル推論問題は、Ｍに関し、有限の L_0 -完全公理化を見つけることにある。

モデル推定問題を解くアルゴリズムは、モデル推論アルゴリズムと呼ばれる。GoldとBlums [Gold 65, Blums & Blums75]の定義と合わせる為に、そのようなアルゴリズムの概念をもっと正確にする。

モデルＭの枚挙は、無限の並び F_1 ，F_2 ，F_3 ，……である。ここで F_i は、Ｍについての事実である。また、L_0 のどの構文αも、$i > 0$ とする事実 $F_i = <\alpha, V>$ が存在する。モデル推論アルゴリズムは、与えられた観測言語 L_0 に対し、モデルの枚挙を読み込み、即ち、１度に１つの事実を読み、ときどき、仮説言語 L_h の構文の有限集合（アルゴリズムの推測とよばれる）を出力するアルゴリズムである。

モデル推論アルゴリズムが実際にいくつかの推論を出力し、異なる推論を二度と出力しなければ、モデル推論アルゴリズムは、モデルの枚挙に与えられた極限に収束していると言われる。モデル推論アルゴリズムが極限でモデルＭに同一視すると言われる為には、Ｍのすべての枚挙上で、そのアルゴリズムがＭに関し L_0 完全公理化となる推測に収束することである。モデルＭの枚挙を与えると、Ｍに対してOracleを模擬できる。また、逆もまた真である。それは、我々がこの問題の抽象化の中でこれらの概念の区別を行なっていないことによる。しかしながら、以下で示すが、実験を行う能力は、推論過程で顕著な高速化を結論する。それは、具体的アルゴリズムは、通常、Ｍに対するOracleが与えられていると仮定されているからである。

2．2　許容的な要請

モデル推論アルゴリズムにより解かれるモデル推論問題の為に、許容的な要請がL_0とL_hに行なわれなければならない。それは本質的に、L_0が、任意の為の理論を反ばくするのに十分なる情報を含んでいるということである。これは、L_0が完全になる事を試行している。この許容的な要請は、Popperの方法論的要請を表わしている。彼によれば、理論は、事実により、反ばくできるといっている。

定義2．2：　L_0を観測言語とし、L_hを仮説言語とする。
$<L_0, L_h>$が許容的であるためには、Lのすべてのモデルとすべての$T \subset L_h$に対し、$\{\alpha \in L_0 \mid T \vdash \alpha\} = L'_0$で、TがMで真になるTを導出することにある。

許容的な要請により、L'_0を証明するL_hで偽となる理論は、観測言語L_0で、虚偽性に対する証拠をもつ。これにより、$T \vdash L'_0$かつTがMで偽であれば、Mで偽となる$\alpha \in L_0$が存在する。ここで、$T \vdash \alpha$である。この性質がL_0とL_hの許容性を導出することを調べるために、この性質が成立することを前提にして$T \subset L_h$がMで偽であるとしてみよう。このとき、Mで偽なる$\alpha \in L_0$に対して$T \nvdash L'_0$又は$T \nvdash \alpha$のどちらかが成立する。両方とも、許容的な要請を満足するが、それらは、L_0とL_hの許容性を導出する。許容的な要請は、観測言語と仮説言語を一緒にする。また、それらの表現力の差は、そのような方法で縛られるべきなので、成功するすべての理論（例えば、真の観測文を全て導き、1つも偽の観測文を導かない）は、また真になるべきである。許容的な要請から次の事が言える。もし、L_0とL_hが許容されると、$L_0 \subset L'_0 \subset L'_h \subset L_h$をみたすすべての$L'_0$と$L'_h$は、また許容される。

観測と仮説について、興味ある2組の言語がある。1つの組は、観測言語がLのground atomsであり、仮説言語がLのホーン構文となる組である。この組に対して、MのL_0 -完全公理化は、atomic -完全公理化と呼ばれる。他の1つの組は、観測言語が、Lのground unit 構文であり、仮説言語がLとなる組である。この組に対し、L_0 -完全公理化は、ground -完全公理化と呼ばれる。これらの組の許容性を見る前に、組$<\{\text{atom of } L\}, L>$が非許容的である事を示す。陳述的例は、十分ある。Lは、3つのアトムP，Q，Rを含むとする。PとQが偽で、Rが真になるLのモデルMで選ぶ。このとき、$L'_0 = \{\{R\} \leftarrow\}$であり、$T = \{\{R\} \leftarrow, \{P, Q\} \leftarrow\}$が$L_0$を制限した結果は、$L_0$に等しいが、Tは、Mで偽である。この例と、次の二つの定理は、ホーン節とatomic -完全公理化の親密な結合の中でいくつかの洞察力を与える。
また［Van Emden & Kowalski　76］を、この結果の議論として参照されたい。

定理２．３：　Ｌを１言語として
L_0 ＝｛Ｌのground atoms｝，L_h ＝｛Ｌのホーン構文｝とする。このとき、組 $<L_0, L_h>$ は、許容的である。

証明：$T \subset L_h$ は、Ｍで偽とし、$T \vdash L_\square$ を仮定する。L_0 は、Ｔの虚偽性に対する証拠をもつとする。
Ｔは、偽なので、Ｍで偽なる構文 $P = \{P\} \leftarrow \{Q_1, Q_2, \cdots\cdots, Q_n\} \in T$ が存在するが、これは、ＰがＭで偽なる（定義４．１を、参照）ground instance $P' = \{P'\} \leftarrow \{Q_1', Q_2', \cdots\cdots, Q_n'\}$ をもつことを導く。$T \vdash L_\square$ なので、$T \vdash Q_i'$ (for $1 \leqq i \leqq n$) ということになる。
$P \in T$ から $T \vdash P'$ が導かれる。Ｐ’は、偽の観測文である。そのために、この組は、許容される。 ∎

定理２．４：　Ｌを１階述語の言語とする。
L_0 ＝｛Ｌのground unit 構文｝，L_h ＝Ｌのとき、組 $<L_0, L_h>$ は、許容的である。

証明：類似的証明を行う。$T \subset L_h$ が、Ｍで偽であり、$T \vdash L_\square$ を仮定する。Ｔは、偽であるので、Ｍで偽の構文
$P = \{P_1, P_2, \cdots\cdots, P_m\} \leftarrow \{Q_1, Q_2, \cdots\cdots, Q_n\} \in T$
が存在する。これは。Ｍで偽のground instance
$P = \{P_1', P_2', \cdots\cdots, P_m'\} \leftarrow \{Q_1', Q_2', \cdots\cdots, Q_n'\}$
をもつ。この構文は、$Q_1', Q_2', \cdots\cdots, Q_n'$ が、真で、$P_1', P_2', \cdots\cdots, P_m'$ が、偽のときだけ偽になる。$P \in T$ なのでＴは、矛盾する。これにより、Ｔは、特殊な場合の $T \in \square$ を含めて、Ｌのすべての構文を導出する。□は、偽の観測文である。 ∎

２．３　理論項の組み込み

許容的な要請は、モデル推論問題の解決性に対して、必要ではあるが十分条件でない。例えば、我々の領域が算術式の標準的モデルであれば、１階述語の言語Ｌは、以前に述べた算術式の述語であり、$L = L_h = L_0$ を選ぶと、組$<L_0, L_h>$は、許容的である。ゲーテルの第２不完全定理は、Ｌの中に有限の矛盾した公理は存在しないといっているが、その公理は、算術式として真なるＬのすべての構文を導出する。

算術式の場合、仮説言語Ｌを高めることは、このモデル推論問題を解くことにならない；しかしながら次のような事情がある。問題の非解決性は、仮説言語の表現力を弱める事になる。例えば、Ｌは、単一述語記号in（Ｘ）をもつFigure1-2 に述べた２進ストリングの言語であるとする。また、推論される未知の集合は、０，１の回分であるとする。ストリングＳが、それ自身の逆であれば、Ｓは、回分である。

回分の集合は、秩序立っていないので、Angluin とHoover[Angluin & Hoover 80] は、すべての真なる回文ＳによりIn（Ｓ）を導出する有限の構文集合が存在しないと結論している。

直観的に言えば、回文の概念を理解するためには、まずストリングの反転の概念を知る必要がある。また、ある方法で、この概念を手にしたり、その操作をシュミレートしなければ、回文と非回文の例から回文の概念を学習できない。仮説の構文を目的として使えるが、その世界で直接観測できない概念を示す項は、科学の理論がもつ哲学と呼ばれている。たとえば、理論項としてappendとreverseの述語を仮説言語L_hに含めても、回文のatomic完全公理化を推論する仕事が、すぐに解けることにならない。しかし、容易に以下の公理を満たす。

in（Ｘ）←reverse （Ｘ，Ｘ）

また、reverse は、Figure1-3 に示すようにappendを使って公理化され得る。

しかしながら、我々がその世界で見るすべてのストリングが、いくつかの未知の集合の入出力にあるストリングであるとするならば、appendとreverse に関する公理を例から推論するための我々の方式は１つも存在しない。それを設定するために、我々の過程は、少なくともappendとreverse に関してもくろまれた解釈が理論的概念として推論アルゴリズムで知られているか構成されているかということである。モデル推論問題に理論的述語の概念を組み込ませる方法は、推論アルゴリズムとして知られているいくつかの固定的な解釈をもつ理論的概念を要求する事にある。この解釈は、未知モデルＭで成立すると仮定する。ここで、このアイデアを公式に展開しないだろう。しかし、その関連は、科学的過程の反ばく性の論議で行なわれる（４．２節）。それは、モデル推論問題の紹介になる。

３．モデル推論問題の複雑さ

モデル推論問題に密接に関係した２つの帰納的推論問題が、文献の中に有益な注意を与えてきた：１つは、再帰的に数え上げられる集合を数え上げる為のプログラムを帰納的に推論するための問題である。この集合には、各要素に、その集合の入／出力情報が与えられている［Gold 65,Gold 67,Klette and Weihagen 80］：他は、帰納関数を計算する為のプログラムを帰納的に推論する問題である。その関数［Gold 65,Blum & Blum 75］の値が、与えられている。Gold[Gold 65]は、再帰的に、数え上げられる集合または、すべての帰納関数に対して、これらの問題を解くための一般解が、存在しないことを示した。Blumたち[Blun & Blum 75]は、後半の問題を解くためのアルゴリズムを示しており、同一視される関数は、ある完全な測定下で十分に計算することができるとしている。また、たしかな過程の下で、彼らが示したアルゴリズムは、任意の推定アルゴリズムの能力の上限をいっているとしている。Blumたちの結果は、極限で帰納的に推論される対象の自然なクラスは、複雑なクラスであると提案している。この節では、モデル推論問題に似た結果を得る。ある帰納的関数ｈに対しh-easyモデルのクラスが、定義される、また任意のh-easyモデルを極限で推論する簡単で効率的なアルゴリズムが、述べられる。もし、帰納的推論アルゴリズムが、単純な充足的要求を満たせば、即ち、現在知られている事実を、その現在の推測がいつも説明しているならば、そのアルゴリズムは、ある帰納関数ｈとする、h-easyモデルだけに同一視することができる。これは、帰納的推論アルゴリズムの能力の上限である。

枚挙型の推論アルゴリズムは、１階述語がもつ意味と構文について、十分な優位性をもっていない。また、類似の枚挙型アルゴリズムは、ほとんどどんな他の計算モデルの中でもインプリントされ得る。ロジックの性質で優位性をとる推論過程を高速化する方法が、以下の節に示される。

３．１　枚挙型アルゴリズム

ある許容対を$<L_0, L_h>$とする。$\alpha_1, \alpha_2, \alpha_3, \cdots\cdots$を$L_0$の全ての文に対して定められた効果的な枚挙であるとする。$T_1, T_2, T_3, \cdots\cdots$を、Ｌの有限集合として定められた効果的な枚挙であるとする。Ｔが、ｎ回の導出段階又はそれ以外でαを導出できる｛できない｝事を示すために$T \vdash_n \alpha$｛$T \not\vdash_n \alpha$｝を利用する。また、任意の有限の構文$T \subset L$と任意のｎに対し、ｎ回の導出段階でＴから導出される構文の集合は、計算可能で有限であると仮定される。我々は、また導出手続きは、単調である。即ち、任意のｑ，ｐ，Ｔに対して、$T \vdash_n p \rightarrow T \cup \{q\} \vdash p$であると、仮定する。このような証明手続きの例として、分解証明法[Robinson 65]がある。

以下のさらに具体的な議論で、分解証明法は、我々の証明手続きであると仮定する。しかし、この選択は、本節のさらに抽象的な結果にとって、重要でない。アルゴリズム１は、次の簡単なアイデアをインプリメントしている：既知の事実と一致する推測を考えている間、推測が成立している。一度、推測が成立しないことがわかると、考えられる次の推測に対する枚挙をさがす。推測が事実と一致するか否かという判断は、一般に非決定的である。その時に、先験的なある境界を、選択すると、任意に与えられた事実で、この判定を行うことになるだろう。

アルゴリズム１が極限で推論するモデルの集合を特徴ずけるためにいくつかの完全な論理的道具と定義を必要とする：これらのもっと完全な説明に対しては [Hachtey & Young 78] を見よ。公理T_kの各集合と階段表示関数Φ_kを関連ずける。$\Phi_k(i) = \min\{n \mid T_k \vdash_n \alpha_i\}$．この関数は、部分帰納関数である。また読者は、階段表示関数Φ_1，Φ_2，Φ_3，……の集合が、次のように定義される特性関数ψ_1，ψ_2，……の集合で、複雑さの尺度を構成していることがわかる。

$$\psi_k(i) = \begin{cases} T_k \vdash \alpha_i \text{ ならば、} 1。 \\ \text{その他は不定。} \end{cases}$$

アルゴリズム１：枚挙型のモデル推論アルゴリズム．

hを、全帰納関数とする。
S_{false}を{□}、S_{true}を{}、そしてkを0、とする。
repeat.
　次の事実$F_n = <\alpha, V>$を、読み込む。
　αを、S_Vに加える。
　while. $T_k \vdash_n \alpha$となる$\alpha \in S_{false}$が、存在するか
　　または、$T_k \nvdash_{h(i)} \alpha_i$となる$\alpha_i \in S_{true}$が、存在する。 do.
　kを、k＋1とする。
　output T_k.
forever.

L_0 の結論が、ある全帰納関数hを法として、容易に得ることができれば、すなわち、$T \vdash \alpha_n$ を満すようなほとんどすべてのn＞0に対し、$\Phi_k(n) \leqq h(n)$であれば、有限集合としての、構文Tは、h-easyであるといわれる。T_k を、Mでh-easyなL_0 -完全公理化とするk＞0が、存在すれば、モデルMは、h-easyである。もし、Mが、h-easyであれば、アルゴリズム1は、極限で、Mを推論できる。すなわち、ある有限個の事実をたしかめた後、このアルゴリズムによる現在の推測は、モデルMのL_0 -完全公理化になる。また、この推測は、その後も、変化しない。いいかえると、

定理3．1： hを、ある全帰納関数とし、MをLのh-easyモデルとする。このとき、アルゴリズム1は、極限でMと同一視する。

この定理を証明する為に、次の補題を必要とする。モデルが有限のL_0 でひどく（複雑に）ふるまう有限のL_0 -完全公理化をもつならば、異例な場合をすべて改善し、他の場合を少なくとも良くするようなMの有限Lo 完全公理化が他に存在する。

補題3．2： Mを、ある全帰納関数hのh-easyモデルとする。このとき、次のようなkが存在する。

1．T_k は、Mで真である。

2．α_n は、Mで真なるすべてのnに対して$\Phi_k(n) \leqq h(n)$である。

証明：Mは、h-easyであるので、T_{k1}はMの有限なL_0 -完全公理化となるk1であり、ほとんどすべてのn＞0に対し、$\Phi_{k1}(n) \leqq h(n)$である。この二条件を満足するkを見つけるために、すべてのi＞0に対し、α_i がMで真であり$\Phi_{k1}(i) > h(i)$が成立する構文α_i を、追加することによって、T_{k1}を修正している。このように定義された構文の有限集合は、$L_0 \subset L_h$ なのでL_h に存在する。k_2 を構文のこの集合に関するインデックスとしよう。T_{k1}は、MのL_0 完全公理化であり、T_{k2}は、Mで真なる構文を追加する事によって、T_{k1}から得られると、T_{k2}は、またMに関しL_0 完全公理化である。$T_{k1} \subset T_{k2}$からすべてのnに対し、$\Phi_{k2}(n) \leqq \Phi_{k1}(n)$であることがわかっている。∎

さらに、関連する構文がT_{k2}にあるので、$\Phi_{k1}(i) > h(i)$とするすべてのiについて、$\Phi_{k2}(1) = \uparrow$である。

定理３．１の証明：ｋ０を、以下のような最小のｋとする。

１．T_k は、Ｍで真。

２．α_n は、Ｍで真なるｎ＞０に対しΦ_k（ｎ）≦ｈ（ｎ）。

そのようなｋは、補題３．２により存在する。

ｋ＜ｋ０とする任意のｋは、少なくともこれらの条件を満足しない。

もし、T_k が条件２を満足しなければ、Ｍで真なるいくつかのα_n に対して、$T_k \not\vdash_{h(n)} \alpha_n$ のとするｎ＞０が存在する。T_k は、条件２を満すが、条件１を満さなければ、L_h とL_0 の許容的な要請により、Ｍで偽なるいくつかのα_{n1}が存在する。ここで、$T_k \vdash \alpha_{n1}$である。$T_k \not\vdash_h \alpha_{n1}$から、最小のｎ≧ｎ１とするｎを選択してみる。どちらの場合でも、ｎ番目の事実を読むと、white-loopの条件が満足され、ｋの値が増加する。ここで、有限の事実をたしかめると、アルゴリズム１は、ｋ＜ｋ０に対する推測T_k のすべてを捨てるだろう、一方、このアルゴリズムがいつもT_{ko}を推測するならば、種々の推測を決して再び出力しないだろう。その理由は、T_{ko}が、Ｍで真で、ｈ（ｉ）ステップでＭで真なるすべてのα_i を、導出するからである。そのために、有限の事実を説明した後、アルゴリズムは、T_{ko}を推測し、二度と、異なる推測を出力しないだろう。∎

３．２　充足的なアルゴリズムの能力限界

アルゴリズム１は、任意の帰納関数ｈに対する、すべてのh-easyモデルを、極限で、同一視できる帰納推論アルゴリズムが存在することを、示している。本節では、帰納推論アルゴリズムが充足的であるならば、ある定められた帰納関数ｈに対するh-easyモデルにだけ極限で同一視が可能であることを示す。これにより、任意のこの様なアルゴリズムの能力の上限を確立する。

定義３．３：　$F_i = <\alpha_i, true>$，１≦ｉ≦ｎのすべての事実に対し、Ｔがα_i を導出すれば、事実$F_1, F_2, \cdots, F_n$ に関し、構文Ｔは、充足的であると言われる。

記述されているアルゴリズム１は、充足的である。この性質は、推論アルゴリズムに関し、妥当な要求であるかのように思われる。それは、Blums の論文で、それを満さないいくつかの強力な推論アルゴリズムが、述べられている事を知っての事である。次の定理は、充足的な帰納アルゴリズムを要求すれば、h-easyモデルがそれ自身である事を示している。

定理３．４：　Ｉを、言語Ｌに対する充足的な帰納アルゴリズムであるとする。このとき、Ｉで、一様な全帰納関数ｈが存在する。すなわち、Ｉが、ＬのモデルMに同一視するならば、Mは、h-easyである。

証明：下記の補題３．５により、Ｉで一様な充足アルゴリズムＩ’が存在する。Ｉ’は、Ｉと同じくらい強力であり、すべての事実をいくつか読むだろう。

Ｓ（ｎ）を形成$\sigma = (<\alpha_1, V_2>, <\alpha_2, V_n>, \cdots, <\alpha_n, V_n>)$をもつ矛盾のない有限列とする。ただし、空節は、含まない。すべての$\sigma \in S$（ｎ）に対し、この順序で、σの事実を読んだ後と、ｎ＋１番目の事実を読み込む前にＩ［σ］を、最後の推測アルゴリズムＩ’の出力のインデックスであると定義する。Ｉ’はいつかすべての事実を読むので、Ｉ［σ］は、妥当な定義であり、計算可能である。

Ｉ’は、充足的であるので、すべての$\sigma \in S$（ｎ）と、σのすべての事実$F_i = <\alpha_i, \text{true}>$に対し、いくつかの有限ステップで$k = I[\sigma] \rightarrow T_k \vdash \sigma_i$を満す。また、そのために、$\Phi_k$（ｎ）が定義され、$V_i = \text{true}$とするすべてのｉに対し有限になる（$0 \leq i \leq n$）。我々は、次のような複雑な境界ｈを選択する。

$h(n) = \max\{\Phi_k(n) \mid k = I[\sigma], \sigma \in S(n), V_n = \text{true}\}$

Ｓ（Ｎ）の並びには、空節を含まないので、すべてのｎに対する$\sigma \in$（ｎ）が、存在する。これにより、ｈは、全帰納的である。

Ｉ’が極限で推論できるすべてのモデルは、h-easyである事を示す。Mを、そのモデルとし、その列$<\alpha_1, V_1>, <\alpha_2, V_2>, \cdots$にＩ’を、適用してみよう。Ｉ’が$M$で同一視できるならば、十分に大きなｎに対しｎ個の事実を読んだ後、MのあるL_0完全公理化T_iに収束するだろう。ｈの定義により、十分に大きなｎに対し、Φ_i（ｎ）$\leq h$（ｎ）であり、Mは、h-easyになる。■

補題３．５：　Ｉを、十分に帰納的な推論アルゴリズムとする。そのとき、Ｉで、一様かつ充足的な帰納推論アルゴリズムが存在する。即ち、Ｉ’は、Ｉと同じくらい強力であり、すべての事実をいつかは読む。

証明：この補題は、Minicozzi のユニオン理論から結論されるが、[Blum & Blum 75] にある定理３の証明と、変わらない。■

４．決定的な実験(Crucial Experiments) による仮説の反ばく

アルゴリズム１の実行不可能性に関する主な原因は、その大局的性格にある。ある一組の構文が、モデルの公理化でないことがわかれば、単にそれをすて、次の発見的推測を見つけるまで、有限なすべての構文の組を調べる。本節では、増殖的推論アルゴリズムを開発するために、解かなければならない最初の問題を提供する。即ち、現在の推測が非常に強いこと（例えば、偽の観測結果を導く）が分ったとき、何をするべきかということである。この場合、少なくとも仮説の一つが偽であると結論できる。あるアルゴリズムは、偽の観測文から推測の中で偽なる仮説を、検出できるように開発されている。このアルゴリズムは、矛盾探索アルゴリズムと呼ばれており、仮説が偽になった原因をさかのぼって推測と事実の矛盾をたどる事ができる。このアルゴリズムと、科学理論の反ばく性に対する哲学的問題の関連性が議論される。また、ロジックプログラムをデバックする為のアプリケーションが示される。

４．１　矛盾探索アルゴリズム

決定的な実験は、競争的な科学理論を選択する可能性をもつ実験である。良い結果をもつ決定的な実験は、その予測の１つと反対の例を与える競争理論の少なくとも１つを除去する事ができる。科学の進歩における決定的な実験の重要な役割りが、科学哲学をもつ多くの学校によって認識されている。しかしながら、その極限的能力として、強い議論がある。哲学者Pierre Duhemが言うには“実験が我々におしえてくれる唯一のものとしては、使われている命題には、すくなくとも１つの誤りがあるということである；しかし、このエラーが存在する場所は、我々に教えられていない”［Duhem 54］。Popperでさえ科学の進歩に寄与する反ばくの主な仕事を次のように言っている。“我々は、しばしば、理論システムの、ある大きな単位と、ときどきおそらく全システムにだけテストができるということを、認めなければならない。また、それは、この場合、その要素が虚偽性に対し応答的になるべきであるというけわしい推量(a sheer guesswork) である”［Popper 68］。

矛盾探索アルゴリズムは、この制限をある意味で克服することができる。一般に、決定的な実験は、収集された仮説だけを反ばくできるが、矛盾探索アルゴリズムは、連続的に決定的な実験方法を、提案している。この方法は、唯一の偽の仮説を選択する事を保証している。この方法は、エラーが見い出される場所を正確に伝えており、任意の推量を含まない。この連続的方法は、動的である。即ち、実施されるべく実験の順序は、アプリオリに決められない；むしろ、すべての新しい実験の結果は、その後任を、提案する。

最後に、偽の仮説を明白に指摘する。さらに、これらの実験の正しい結果は、反ばく的仮説に反例を与えている。即ち、この仮説構文は、仮説から理論的に導き出され、偽であることが実験的に決定された構文である。

矛盾探索アルゴリズムは、矛盾が、いくつかの仮説と事実の間に導出されると、いつでも適用される。その入力は、仮説と真の観測構文Ｓから空節“□”をもつ順序ずけられた導出木である。

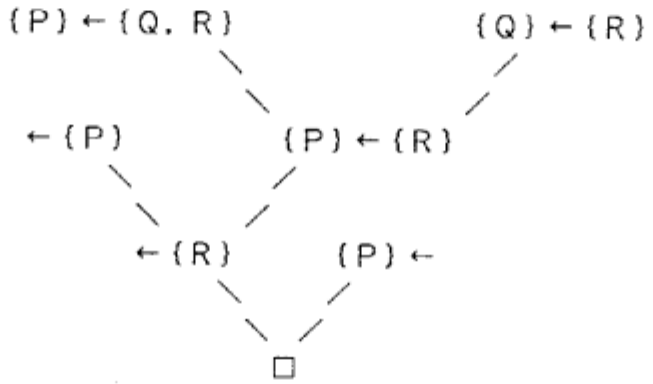
このアルゴリズムには、Ｌのすべてのground atom の真理を告げるＭでのOracleが与えられている。そのアルゴリズムは、Ｍで有限回の実験を行っているが、この実験は、その導出木の深さにより制限され、Ｍで偽の仮説ｐ∈Ｓを出力する。

最初に、このアルゴリズムがFigure4-1 で示される命題計算例で何をやるかを説明する。この例では、

Ｓ＝｛｛Ｐ｝←｛Ｑ，Ｒ｝，｛Ｑ｝←｛Ｒ｝，←｛Ｐ｝，｛Ｒ｝←｝，

である。

Figure4-1 ：命題論理における矛盾のバックトレース．



このアルゴリズムは、ルート□からスタートしている。また、導出されるアトムを反復的に確かめている。もし、アトムがＭで真であれば左の補木をえらび、そうでなければ右の補木を選ぶ。これは、リーフにたどりつくまで行なわれる。リーフにある仮説は、Ｍで偽なることが保証されている。示されている例では、仮説｛Ｐ｝←｛Ｑ，Ｒ｝が、偽なる事を仮定している。これは、ＲとＱがＭで真でＰが偽なる事を意味する。

このアルゴリズムは、RがMで真なるかどうかを、最初にためし、我々の仮定によりそれが真なので左側の補木を選んでいる。次に、Pがそのモデルで偽になっいてることをたしかめ右側の補木を選んでいる。最後に、Qが真であることをたしかめ、左側のブランチを選択したところ、リーフになっていることをたしかめている。Mで偽となっている仮説｛P｝←｛Q，R｝を出力している。

一つの実験に対する答は、次の実験を決定することに注意せよ。また、実験の回数は、まさに到達したリーフの深さであり、これらの実験を結合した結果は、そのリーフでの仮説の反例を構成する。もし、いくつかのリーフが、（仮説よりも）むしろ真の観測構文をもてば、必要とされる実験の回数は、非常に少なくなるかもしれない。

計画的な場合、アトムをすべて確認することによって偽の仮説を、簡単に検出できる。その確認は、Mでそれらの真理を許しているように見える。このだまされやすい手続きは、指数的に必要とされる実験の数を増大するかもしれないが、しかし、原理的には、同じ仕事を行うことができる。述語計算の場合に、この仕事は、より複雑になる。全称的仮説は、無限領域の上に、効果的に確認され得ないが、もしそれが偽ならば、それは偽のground instance をもつ。述語計算の場合に、このだまされやすい手続きは、その少しばかりの領域を増加して、仮説をシステマチックに例証することである。また、仮説の反例、すなわち、それに関する偽の例証が発見されるまで、作成されるground instance （すべてL_0 に存在すると仮定）を、すべてたしかめる。許可されている仮説の少なくとも1つが偽であると、そのアルゴリズムは、停止することが、保証されている。また、そのような仮説は、有限領域でそれを例証したとき、偽の例証をもっている。しかしながら、このアプローチは、明らかに実行不可能である。

1階述語の言語における矛盾探索アルゴリズムは、それに対する反例をシステマチックに構成して偽の仮説を検出するのと、同じアイデアに基づいている。しかしながら、この反例が構成される方法はさらに細く含まれている。このアルゴリズムを記述する前に、導出に関するいくつかの用語を、さらに正確にさせておく必要がある。これらの問題については、ロビンソン［Robinson 65］にならう。置換は、形式V／tの対に観測する有限集合（空も可能）である。ここで、Vは、変数でtはタームである。これらの変数は、どれでも同じでない。
任意の置換θ＝｛V_1／t_1，V_2／t_2，……，V_n／t_n｝と表現sに対し、表現sθは、sで変数V_iの各でき事をタームt_iで置き換えた結果である。もし、置換θに対して、t＝sθであれば、tは、sの例証と呼ばれている。Sが表現の集合であれば、Sθ＝｛sθ | s∈S｝である。もし、二つの構文が、おたがいに例証であれば、それらは変数に名をつけかえることに等しい。また、その他に指摘されなければ、それらを区別しない。

θを、ある置換とする。もし、Sθがある別物であれば、アトムSの有限集合（空ではない）は、θによりユニファイできる。
もし、$\theta_1 = \{t_1/V_1, t_2/V_2, \cdots\cdots, t_n/V_n\}$と$\theta_2$を、2つの置換とすると、$\theta_1 \circ \theta_2 = \theta_1' \cup \theta_2'$が、成立する。ここで、$\theta_2'$は、$\theta_2$のすべての要素の集合である。$\theta_2$の変数は、$V_1, \cdots\cdots, V_n$の中になく、$\theta_1' = \{t_i \theta_2 \mid 1 \leq i \leq n, t_i \theta_2 \neq V_i\}$の中にある。任意のストリングsと2つの置換$\theta_1$と$\theta_2$に対し、等号$(s\theta_1)\theta_2 = s(\theta_1 \circ \theta_2)$が成立することが示される。もし、$\theta$がSのユニファイアであれば、置換$\theta$は、Sに関し最も一般的なユニファイアである。また、Sに対する任意の他のユニファイアθ_1に対し、$\theta = \theta_1 \circ \theta_2$である様な置換$\theta_2$が存在する。もしそれが存在するとき、Robinsonは、任意の集合に関する最も一般的なユニファイアを計算するアルゴリズムを、述べている。
PatersonとWegman[Paterson & Wegman 76]は、linear time で操作するアルゴリズムを述べている。

定理4．1　A←B，C←Dを、Lの2つの構文とする。　ただし、$R_1 \subset$ B，$R_2 \subset$Cである。もし、集合R_1とR_2が、最も一般なユニファイアθをもてば、(P)$=R_1\theta = R_2\theta$であり、このとき[A∪(C−R_2)←(B−R_1)∪D]θは、A←BとC←Dの、ある導出形である。また、Pは、上で導出されたアトムである。導出形を形成する過程は、左成分にA←B、右成分にC←Dをもつ導出と呼ばれる。

矛盾探索アルゴリズムは、ある反例の断片を作成している。また、そのような反例を見つけるために、関係する実験だけを行っている。これは、そのアルゴリズムをさらに複雑にしている。この述語計算の場合、導出されるアトムは、groundにならないので、MについてのOracleを使って、その真実を直接たしかめることが必ずしもできるとは限らない。この問題の解は、Oracleにそれを与える前に、あるground atom に対するPを例証することにある。Pをいかに例証するかという選択は、任意であるが、しかし一度それが行なわれると、リーフにたどりついた仮説に対する反例を作るために、同じ置換によるすべてのすすんだ実験が行なわれてるべきである。

アルゴリズム２：決定的な実験による矛盾のバックトレース.

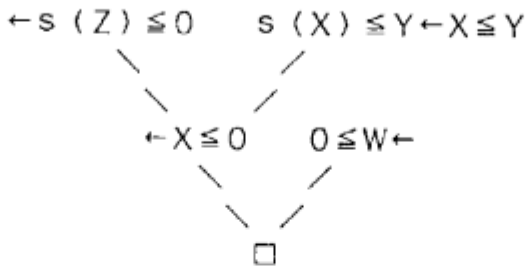
入力：モデルMのあるOracleと、次の性質をもつ構文のOrdered ２進木。
１．ルートは、空節である。
２．すべてのリーフは、仮説又は、観測文のどちらかがMで真であり、２つのリーフは、１つの変数を共有しない。
３．リーフに存在しないすべてのノードは、その子孫に関する２進導出形である。ここで、導出に関し左の子孫は、左の成分であり、右の子孫は、右の成分である。

出力：導出木のリーフに発生し、Mで偽なる一つの仮説。

アルゴリズム：kを0、N_0 をその木のルートとし、θ_0 を{}とする。
while N_k がリーフにならない do.
Pを、最も一般的なユニファイアθをもつN_k で導出されるアトムとする。
$P\theta_k$ をground atom P_k にインスタンシェートする置換θ'、を選択する。
θ_{k+1} を、$\theta \circ \theta_k \circ \theta'$ とする。
P_k が、Mで真であるか否かを調べる。
case P_k が、
真のとき：N_{k+1} をN_k の左子孫とする。
偽のとき：N_{k+1} をN_k の右子孫とする。
kを、k＋１とする。
output N_k .

以下は、モデル推論アルゴリズムによる矛盾探索の利用例である。このアルゴリズムは、自然数で関数 ≦ の有限近似を推論する事を行なっている間、この矛盾探索が行なわれる。ここで自然数は、定数０と後続関数ｓ（Ｘ）で作られる。このアルゴリズムは、仮説 $0 \leqq X \leftarrow$ と $\leftarrow s(X) \leqq 0$ をすでに推測しており、事実 <ｓ（０）≦ｓ（ｓ（０）），true> に出くわしていると仮定する。仮説ｓ（Ｘ）≦Ｘ←Ｘ＝Ｙ を提案すると、仮説 $0 \leqq X \leftarrow$ をもつ構文 ｓ（０）≦ｓ（ｓ（０）） が導出される。しかしながら、この新しい仮説を追加後、Figure4-2 の導出が、また構成される。包含されている仮説が偽になることを検証するために、矛盾探索アルゴリズムを適用してみよう。

Figure4-2 ：述語論理での矛盾のバックトレース.



ルートに導出されるアトムは、 $0 \leqq 0$ である。Oracleは、 $0 \leqq 0$ に対して、'true'を返す。そのため左側の木が選ばれる。
即ち、N_1 が、 $\leftarrow X \leqq 0$ に設定され、θ に $\{X/0\}$ が設定される。
N_1 で導出されるアトムは、 $s(X) \leqq 0$ であり、$(s(X) \leqq 0)\theta_1$ は $s(0) \leqq 0$ である。このOracleは、この確認で'false'を返す。右側の木は、 $\theta_2 = \{Y/0\} \circ \theta_1 = \{X/0, Y/0\}$ で選択され、リーフ $N_2 = s(X) \leqq Y \leftarrow X \leqq Y$ は、偽になることが、検出される。この実験の結果により構成される反例は、 $s(0) \leqq 0 \leftarrow 0 \leqq 0$ であるが、これは、また $N_2\theta_2$ である。仮説 $s(X) \leqq X \leftarrow X \leqq Y$ のすべての具象例が、偽でないことに注意しよう。例えば、置換 $\{X/0, Y/s(0)\}$ は、真なる具象例 $s(0) \leqq s(0) \leftarrow 0 \leqq s(0)$ を生ずる。

このアルゴリズムの正しさを示す前に、実験の結果が、仮説に対する反証を、いかに構成できるかをさらに正確に定義しよう。
$A'\theta \subset A$ かつ $B'\theta \subset B$ あるいは要するに、$(A' \leftarrow B')\theta \subset (A \leftarrow B)$（5．3節を見よ）となるような置換$\theta$が、存在すれば、構文 $A' \leftarrow B'$ は、構文 $A \leftarrow B$ を包含する、と言う。ground atom $P_1, P_2, \cdots\cdots, P_k$ が確められると仮定してみよう。BをMで真なるアトム $\{P_1, P_2, \cdots\cdots, P_k\}$ の部分集合とし、AをMで偽なる部分集合とする。このとき、ground sentence $A \leftarrow B$ は、Mで偽であり、それを含むすべての構文に類似している。そのような場合、$A \leftarrow B$ は、$P_1, P_2, \cdots\cdots, P_k$ を確認した成果から構成されたθによるpの反例として参照される。

補第４．２：　Mをしのモデルとし、Tをしの構文から空節をもつOrdered２進導出木とする。もし、アルゴリズム２をTに適用し、構文N_kが、ground atom P_1，……，P_kを確めた後にたどりつくならば、N_kは、Mで偽であり、これらの確認の成果は、θ_kによるN_kの反例を構成する。

証明：この証明は、kの帰納と、多くの確認により存在する。もし、k＝０ならば、N_0＝□である。これば、Mで偽でありθ_0＝｛｝である。□θ_0の実験なしに構成される反例は、また空節である。

ground atom P_1，……，P_kを確認するアルゴリズム２は、ノードN_kにたどりつくとし、この確認の成果は、置換θ_kにより構文　N_k＝（A'←B'）に対する反例　A←B　を構成すると、帰納的に仮定しよう。PをN_kで導出されたアトムとしよう。N_kの左側の子孫は、$A_1 \leftarrow B_1 \cup R_1$　であり、N_kの右側の子孫は、$A_2 \cup R_2 \leftarrow B_2$　である。集合R_1とR_2は、$R_1\theta = R_2\theta = \{P\}$　かつ　$A' \leftarrow B' = (A_1 \cup A_2 \leftarrow B_1 \cup B_2)\theta$とする、最も一般的なユニファイヤ$\theta$をもつ。

$$A_1 \leftarrow B_1 \cup R_1 \qquad A_2 \cup R_2 \leftarrow B_2$$

$$N_k = A' \leftarrow B' = ((A_1 \cup A_2) \leftarrow (B_1 \cup B_2))\theta$$

$P\theta_k$は、ground atom P_{k+1}に対する、置換θ'により例証され、θ_{k+1}は、$\theta \circ \theta_k \circ \theta'$になり、$P_{k+1}$は、Mで置換されると、仮定する。次のような、２つの場合がある：

場合１：P_{k+1} は、Mで真である。このとき、アルゴリズム２は、$A_1 \leftarrow B_1 \cup R_1$ であるN_k の左側の子孫をN_{k+1} に設定する。帰納的仮定により、$A \leftarrow B$はMで偽である。P_{k+1} は、Mで真なので、構文 $A \leftarrow B \cup \{P_{k+1}\}$ は、またMで偽である。$(A_1 \leftarrow B_1)\theta \subset N_k$ なので、帰納
$(A_1 \leftarrow B_1)\theta\theta_k\theta' = (A_1 \leftarrow B_1)\theta_{k+1} \subset N_k\theta_k\theta'$ が成立する。また$N_k\theta_k\theta'$ 最後にgroundになるので$N_k\theta_k$ に等しい。
$N_k\theta_k \subset (A \leftarrow B)$ であるという事実から、$(A_1 \leftarrow B_1)\theta_{k+1} \subset (A \leftarrow B)$ である。また、$R_1\theta = P$ で $P\theta_k\theta' = P_{k+1}$ であるので、$R_1\theta_{k+1} = P_{k+1}$ である。そのため、
$N_{k+1}\theta_{k+1} = (A_1 \leftarrow B_1 \cup R_1)\theta_{k+1} \subset (A \leftarrow B\{P_{k+1}\})$
が成立する。
一方、N_{k+1} は、θ_{k+1} により $A \leftarrow B \cup \{P_{k+1}\}$ を含む。
そのため、N_{k+1} は、Mで偽であり、P_1，……，P_{k+1} を確めた結果は、置換θ_{k+1} によるN_{k+1} の反例を、構成する。
場合２：P_{k+1} がMで偽である。対照的な議論が、適用される。 ■

次の定理は、補題４．２から、すぐわかる系である。

定理４．３： MをLのモデルとし、TをLの構文による空節をもつOrdered ２進導出木とする。もし、Tの深さがnとすると、Tに適用される。アルゴリズム２は、Mでn回以上の実験をもはや行なわず、Mで偽の仮説pを出力する。 ■

4．2　科学的仮説の反ばく性

反ばく理論で、偽の仮説を検出する事は、むしろ困難な仕事であるかの様に見える。それが、哲学者Pierre Duhemにより、不可能である事が主張された。つまり、“物理学者は実験的確認による孤立した仮説を決して指示しないが、仮説の完全なかたまりだけを指示する”［Duhem 54］。私は、Duhem の言葉をレビューし、矛盾探索アルゴリズムが動作するときの仮定で、それを関係ずけた。本質的に、Duhem の言葉は、唯一に偽にする実験の可能性とは逆に、２つの要素：論理的要素と認識論的要素をもつ。Duhem の言葉の論理的要素は、次のように示される。“実験が、我々におしえてくれる唯一のものは、すくなくとも１つのエラーが存在することである；しかし、このエラーが生ずる場所は、我々に知らせていない、ということである。物理学者は、このエラーが、反ばくを望んでいるその信条に正しく含まれていると、定義するかも知れない。しかし、彼は、他の信条に、それが存在すると確信しているだろうか？　もし、彼がいれば、彼は、自分が利用した他の信条の正当性をすべて絶対的に受け入れる。また、彼の結果の有効性は、彼の信頼の有効性と同じくらい大きい。”

このDuhem の主張は、決定的な実験の真理である。しかしながら、矛盾探索アルゴリズムで示されたように、その事実と理論の間に存在する矛盾が発見されると、アルゴリズム的に次にやるべき実験を定めることができる。それは、その結果が、反ばく理論で偽の仮説を選択することによる。さらにDuhem は、次のようなことを述べている。矛盾が仮説と事実の間に生ずると、ある仮説は、その導出で利用された他のすべての仮説の正当性を正しく受け入れることによってだけ偽である。この矛盾探索アルゴリズムは、次の論証により、この主張を、反ばくしている。ground atom(Variable-free)に関してだけ実際の判断の正当性を許容することにより、仮説の虚偽性を、結論づけることができる。このground atom は、具体的な対象と、でき事を参照する構文である。

Duhem の言葉の認識論的要素は、具体的な対象と、でき事に関する実際の判断が、次のいくつかの論理的仮定を含んでいることにある：“実験を行うか、１つの報告を与える物理学者は、理論の完全なかたまりが正確であることを、絶対的に認識している”理論項の概念を組み合わせるモデル推論問題の拡張は、２．３節で論議した。矛盾探索アルゴリズムは、うまくこの拡張を、利用することができる。非公式に、矛盾探索アルゴリズムが、このモデルでground atom を確める方法は、次のようになる：もし、ためされるアトムの述語が可観測であれば、このモデルで直接、確かめられる。さもなければ、アルゴリズムが理論的述語について、いくつかの組み込み知識をもち、この知識をもってground atom が真か偽かのどちらかを決める事ができると仮定する。どちらの場合も、矛盾探索アルゴリズムは、このモデルで、偽の仮説を導く。さらに、公式的に、この仮定は、L

のすべてのground atom $P = p(t_1, t_2, \cdots\cdots, t_n)$ に対し、$P \in L_0$ であるか、又は、pが、ある固定的解釈をもつ理論述語であるか、のどちらかである。この固定的解釈は、その帰納的推論アルゴリズムで、知られている。

Duhem の言葉の議論をまとめると矛盾探索アルゴリズムは、彼の言葉の論理的要素を反ばくするかのように見える。

1．決定的な実験は、収集されている仮説だけを反ばくすることができる。
2．ある仮説を反ばくするために、他の仮説が真理であることを受容しなければならない。

ここで定義されたように、たとえ、科学理論で使われている項を、理論項と観測項に分解できないとしても、Duhem の言葉の認識論的要素は、今でも成立している。

4．3　応用：ロジックプログラムの虫取り

矛盾探索のアイデアは、ロジックプログラムの虫取りに容易に利用できる。もし、ロジックプログラムが、不正な入出力関係を計算するか、与えられた出力で不正の入力を返すならば、少なくとも次のような節がプログラム中に存在する。その節は、述語の予期された解釈の下で偽である。例として、リストの後続を計算する次のプログラムから、

```
subsequence( [ ], [ ] )
subsequence( L1, [ X | L2 ] ) ←subsequence( L1, L2 )
subsequence( [ X | L1 ], [ Y | L2 ] ) ←subsequence( L1, L2 )
```

我々は、subsequence([a, b, c], [a, a, c, s]) を、計算できる。述語subsequence(X, Y) の予期された解釈の下で（即ち、Xは、Yの後続である）プログラムの第3目の節は偽であり、次の例は、反例である。

subsequence([b], [a]) :－subsequence([ ], [ ])。

この節の‘虫’は、任意の他のリストL2の後続として、任意のリストL1を許すプログラムを発生する。即ち、もはやL1は、L2より長くない。

矛盾探索アルゴリズムをsubsequence([a], [b]) の計算（証明）に利用すると、最初に、subsequence([ ], [ ]) を確認すると、真である事がわかる。次にsubsequence([a], [b]) を確認すると、偽であることがわかる。そして、subsequence の第3番目の節に対する上記の反例を与える。

読者は、subsequence に対する正しいロジックプログラムを見つけることになる。　解は、後で与えられている。

以下では、矛盾探索アルゴリズムをインプリメントしているPrologプログラムを記述する。そこでは、仮説は、ホーン節であり、観測言語は、すべてアトムであり、Oracleはユーザである。また、ユーザが、虫のあるquicksort プログラムの虫取りをしようとしているときの注釈付きリストを示す。

矛盾探索アルゴリズムにより利用されている導出の書き換え収集法は、Green Green 69] により提案されたのが、はじまりである。彼のゴールは、証明の中で使われた仮説と、置換の収集からこの節を証明する事により、ロジカルな構文として形成された質問に対する応答を構成する事にあった。Prologでは、次の点を除くと、非常に一致したアイデアを、用いている。除くべき点は、証明される構文が、　←A　の形をしており、仮説がホーン説になるように制限されている、という点である。

Prologインプリンタは、導出過程としてのすべての置き換えを、自動的に維持しているので、その中で、矛盾探索アルゴリズムをインプリメントするのが極度に簡単になっている。

Figure4-3 は、そのようなインプリメンテーションの全リストである。同じようなインプリメンテーションは、本モデル推論システムに取り入れられている。

Figure4-3:　矛盾探索のPrologプログラム

```
backtrace((P,Q),CE) ←!,backtrace(P,CE),backtrace(Q,CE).
backtrace(P,CE) ←clause(P,Q),backtrace(Q,CE),resolve((P←Q),CE).

resolve((P←Q),CE)←var(CE),!,ask(P,V),(V=false,CE=(P ←Q);V=true).
resolve((P←Q),CE).

ask(P,V)←recorded(fact,(P,V)),!.
ask(P,V)←repeat,write(P),put(63),nl,read(V),(V=true;V=false),
                                              record(fact,(P,V)),!.
```

意図されたモデルMで、偽となるアトムに対して、最初の手続きは、アトムAtomの証明の組み立てを試み、それが真ならば、Atomの導出で使用された為の仮説ＣＥ（反例に対応する）のインスタンスを置換していることになる。読者は、Prologに関する詳細についてPrologマニュアルを参照されたい［Pereira etal. 78］。特に、構文｛Ｐ｝←｛Q_1，Q_2，……，Q_n｝に関する内部表現は、Ｐ←（Q_1，（Q_2，（…，Q_n）…））であり、｛Ｐ｝←は、Ｐ←trueである事に注意せよ。

backtrace の手続きは、単純なPrologインプリンタであるが、このインタプリンタは、ある節の導出がある毎に増大する。

backtrace(Atom,CE)の呼び出しは、まず、アトムAtomの証明を最初に組み立てている。これは、ある証明が存在し、証明の再帰的な組み立てから‘popping’backしている途中、節Ｐ←Ｑを証明するこめにresolve(P ←Q,CE) を呼び出す時にかぎる。導出に関係する基礎的なロジックは、すべて巻きこんである。

この決定的なアイデアは、次のようなものである。

Ｐが偽になるresolve((P←Q),CE)を最初に呼び出すと、ＣＥに節　Ｐ←Ｑ　がユニファイされる。　このアイデアの正当性は、次のように帰納的に論じられる：もし、Ｑ＝trueでＰが偽であれば、節Ｐ←Ｑは、偽であり、backtrace は、変数ＣＥに偽節を返す。backtrace が証明の再帰的組み立てからもどってきたときに作られる分解の為の事前呼びだしで、Ｐの上に分解されるアトムがテストされ、真なることが見つけられると、帰納的に仮定しよう。

このとき、条件Ｑは、真である。そのため、もし、Ｐが現在テストされ、偽であれば節Ｐ←Ｑは、偽になる。その結果、変数ＣＥは、　Ｐ←Ｑ　をユニファイし、backtrace は、偽節のインスタンスを返す。補第４．２の証明での、みごとな点は、証明の組み立てと反例の組み立ての中で使われた置換を収集するための必要性によって生じたが、この点は、Prologのユニフィケーション手続きと、共有変数の機能によって、自動的に配慮されている。

以下は、次に示す quicksortプログラムをデバックするために、ユーザが、Figure 4-3のプログラムを実行した実行トレースである。この quicksortプログラムは、最初、いくつかの虫をもっている。それらの虫は、つまらない虫であったり、むずかしい虫であったりする。

```
qsort([X | L],L0)←partition(L,X,L1,L2),
                   qsort(L1,L3),qsort(L2,L4),append(L3,L4,L0).
qsort([],[]).

partition([X | L],Y,L1,[X | L2])  ←partition(L,Y,L1,L2).
partition([X | L],Y,[X  | L1],L2) ← X≦Y,partition(L,Y,L1,L2).
partition([],X,[],[]).

append([X  | L1],L2,L3)←append(L1,L2,L3).
append([],L,L).
```

プルグラムの虫取をするために、その意味する解釈を知る必要がある。qsort(X,Y)は、リスト Yがリスト Xをソートしたものであれば、真である。
partition(L,X,L1,L2)は、L1が、X 以下の要素 Lを含み、L2が、X より大きい要素 Lを含めば、真である。以下の実行トレースで、‘|’で、はじまる行は、ユーザにより入力されており、残りの行は、システムの出力である。quicksort プログラムを実行してみよう。

```
| ?-qsort([2,1,2],X).

 X=[]
```

これは、正しくない結果である。そこで、backtrace を呼び出してみる。
このbacktrace は、上記、qsort,partition,appendに対する‘true’または‘false’で答える真偽の質問をする。

```
| ?-backtrace(qsort([2,1,2],X,CE).
 partition([],2,[],[])?
| :true.
 partition([2],2,[],[2]).
| :false.

 X=[],
 CE=(partition([2],2,[],[2])←partition([],2,[],[]))
```

backtrace は、反例として、partition の最初の節を、選出した。この節は、その帰結部が偽のあいだ、その条件部が真であるので、偽である。X>Y の判定が考慮されていないという明白な誤りがある。この判定をつけ加えて、この節を、修正すると、

```
partition([X | L],Y,L1,[X | L2])←X>Y,partition(L,Y,L1,L2).
```

そして、その新しいプログラムを、実行してみよう。

```
| ?-qsort([2,1,2],X).

 X=[]
```

それは、今までどおり偽の答を返すので、ふたたび、backtrace を呼び出してみよう。

```
| ?-backtrace(qsort([2,1,2],X), CE).
 partition([2],2,[2],[]).
| :true.
 partition([1,2],2,[1,2],[])?
| :true.
 partition([],1,[],[])?
| :true.
 partition([2],1,[],[2])?
| :true.
 qsort([],[])?
| :true.
 append([],[],[])?
| :true.
 qsort([2],[])?
| :false.

 X=[],
 CE=(qsort([2],[])←partition([],2,[],[]),
                   qsort([],[]),qsort([],[]),append([],[],[]))
```

今、CEは、反例として、qsort それ自身を、制限している。qsort([2],[]) に対するすべてのサブゴールが、ただしい答を返したことは、明白である。我々は、そのpartition が、最終結果で実施されるその要素を含めることを、単に、わすれたのである。すなわち、appendの呼び出しを修正すれば、それが可能になる。

```
qsort([X | L],L0)←partition(L,X,L1,L2),qsort(L1,L3),
                   qsort(L2,L4),append([X | L3],L4,L0).
```

qsort([],[])は、いつでも、そのアルゴリズムによりテストされるが、最初のテスト後、その事実 <qsort([],[]),true> がデータベース内に記録されるので、ユーザは、一度だけ、その質問になやまされる事に、注意しよう。虫取りが進行するので、データベースは、虫取りが実施されている手続きについて、多くの事実をもつようになり、その過程のなかでのユーザの干渉は、だんだん少なくなる。このアルゴリズムの増殖的ふるまいは、比較的、大きなシステムの虫取りをするのに、非常に有用である。ふたたび、qsort を実行してみよう。

```
| ?-qsort([2,1,2],X).

X=[]

yes
| ?-backtrace(qsort([2,1,2],X),CE).
append([2],[],[])?
| :false.

X=[],
CE=(append([2],[],[])←append([],[],[]))
```

appendの中で明白な虫が、発見されている。プログラムは、全体として、デバックされ、そのメインの手続きをデバックするまえに、そのサブの手続きをデバックを終了する必要がない。appendの最初の節を、訂正してみよう。

```
append([X | L1],L2,[X | L3])←append(L1,L2,L3).
```

再度、qsort を、実行してみよう。

```
| ?-qsort([2,1,2],X).

X=[2,1,2]

yes

| ?-backtrace(qsort([2,1,2],X),CE).
append([2],[],[2])?
| :true.
qsort([2],[2])?
| :true.
append([],[2],[2])?
| :true.
append([1],[2],[1,2])?
| :true.
```

```
qsort([1,2],[1,2])?
| :true.
append([1,2],[],[1,2])?
| :true.
append([2,1,2],[],[2,1,2])?
| :true.
qsort([2,1,2],[2,1,2])?
| :false.

X=[2,1,2],
CE=(qsort([2,1,2],[2,1,2]) ←partition([1,2],2,[1,2],[]),
                            qsort([1,2],[1,2]),qsort([],[]),
                            append([2,1,2],[],[2,1,2]))
```

ここで、何が誤まりであるかを、みつけることは、少しむずかしいが、丁度、訂正したqsort の節をみると、その誤りの訂正が明白になる。そのpertition を実行するときのその要素は、その最後の結果に追加されるが、まちがった立場では、その要素は、それよりも大きい方のリストではなく、小さい方のリストのヘッドにある。これは、次のように、容易に訂正できる。

```
qsort([X | L],L0)←partition(L,X,L1,L2),qsort(L1,L3),
                   qsort(L2,L4),append(L3,[X | L4],L0).
```

再度、qsort を実行してみよう。

```
| ?-qsort([2,1,2],X).

X=[1,2,2]

yes
| ?-qsort([5,2,45,3,24,543,2,3,1,],X).

X=[1,2,2,3,3,5,24,45,543]

yes
```

それは、正しく動作している。虫取り後、その最終的なquicksort プログラムは、次のようになる。

```
qsort([X | L],L0)←partition(L,X,L1,L2),
                    qsort(L1,L3),qsort(L2,L4),append(L3,[X | L4],L0).
qsort([],[]).

partition([X | L],Y,L1,[X | L2])  ← X>Y,partition(L,Y,L1,L2).
partition([X | L],Y,[X | L1],L2) ← X≦Y,partition(L,Y,L1,L2).
partition([],X,[],[]).

append([X | L1],L2,[X | L3])←append(L1,L2,L3).
append([],L,L).
```

実際的な矛盾探索にもとずくいたデバッガーをつくるために、少くなくとも、さらに二つの道具が、必要である。第一番目は、ユーザが入力した誤りを訂正する方式をもつべき道具、である。第二番目は、矛盾探索アルゴリズムが、あるプログラムの呼び出しに成功する場合にだけ適用され得る道具、である。しかし、そのプログラムは、誤った答をもっている。もし、その呼び出しが、失敗したならば、他のアプローチが必要である。次の節では、それが失敗した場合、いかにして、ロジックプログラムに節集合をつけ加えるかについて、一つのアプローチを提案するが、新しい節をつけ加えるよりも、既存の節の修正を基本とする他のアプローチは、このようなデバッキングの仕事にとって、より実際的であるかもしれない。

5．反ばくされた仮説の精密化

1．2節で述べた2つの問題を思い出してみよう。矛盾探索アルゴリズムは、非常に強い推測の中から、偽の仮説を検出することにより最初の1つを解いている。そのような仮説を推測から取り除くと、非常に弱い推測になるかもしれない；この節では、その推測をいかに強くするかという問題を述べる。この仕事のために、L_h の構文上に精密化演算子（refinement operator ）を考案する。直観的に、精密化演算子は、反ばく仮説に対して論理的に比較的弱い発見の置き換えをする。次の方法で、増殖的な帰納推論アルゴリズムを利用することができる：ある仮説に対する反例が発見されたときはいつでも、その仮説の除去により、反ばくされた推測を修正する。もし、結論された推測が、弱すぎるならば、他の仮説を付加し、その推測を強くする。ここで、他の仮説は、この精密化と、以前に反ばくされた仮説から選択される。

以下では、精密化の概念と、精密化演算子を明らかにする。精密化演算子の完全性は、与えられた仮説言語に対して定義されている。また、いくつかの例が、種々の精密化演算子として与えられるが、この演算子は、仮説言語の種々のクラスに対して完全である。

L_h の構文で精密化演算子により導かれる精密化グラフの特性が述べられる。任意の1階述語言語Lに対して、完全である最も一般的な精密化演算子についても述べられる。

5．1　精密化演算子

ある構造的な複雑度を表わす尺度を、仮定する。この尺度は、Lの構文から自然数へ対応ずける関数であり、すべての$n>0$に対し、尺度nの構文は、有限であるという特性をもつ。
構文Sの任意の集合と任意の$n>0$に対し、集合$\{p \in S \mid size(p) \leqq n\}$を$S(n)$と定義する。

定義5．1：　Lを、1階述語とし、pとqをLの構文とする。このとき、pがqを導出しsize(p) ≦size(q) ならば、qはpの精密化であるという。

定義5．2：　精密化演算子ρは、Lの構文から、それらの精密化に関する部分集合への写像である。したがって、任意の$p \in L$と任意の$n>0$に対し集合$\rho(p)(n)$、すなわち、size≦nの構文として制限される集合$\rho(p)$は、計算可能である。

L上の精密化演算子は、L上の半順序$\lesssim_\rho$を導出するが、この半順序は、最少要素として空節をもつ：$p_{i+1} \leftarrow \rho(p_i)$を満たす有限の節

$p = p_0, p_1, \cdots, p_n = q$は、

有限の全 -ρチェイン（finite total ρ-chain）と呼ばれる；もし、pからqへの有限の全ρチェインが存在するならば$p \lesssim_\rho q$とする；もし、$p \lesssim_\rho q$かつ$q \lesssim_\rho p$が成立するならば、$p <_\rho q$とする。

関係$\lesssim_\rho$は、構文の集合を一般化する。すべてのq∈Sに対して、$p \lesssim_\rho q$なる構文p∈Tが存在すれば、構文中の二つの集合T，S＜Lに対し、$T \lesssim_\rho S$とする。

任意の構文Pに対し、集合$\{q \in L \mid p \lesssim_\rho q\}$は、$\rho(p)$で示される。集合$\rho(\square)$は、$\rho^*$で示される。

定義５．３：　$S \subset L$を空節□を含む構文の集合とする。このとき、$\rho^* = S$ならば、L上の精密化演算子ρは、Sに対し完全であるといわれる。

精密化演算子に関する本来の用法にもどってみよう：仮説が反ばくされるとき、推論アルゴリズムは、その精密化の中で、仮説の置換を探索する。そのため、そのようなアルゴリズムをもつ精密化演算子は、仮説言語L_hに対して完全になる事が必要とされる。

以下は、モデル推論システムにより利用される具体的な精密化演算子の例であるが、この精密化演算子は、仮説言語の種々のクラスに対し、完全である。

[Reynolds 70] で述べられているように、関係“→”を定義し、

$\rho_1(p) = \{q \mid p \rightarrow q\}$により定義される演算子$\rho_1$を考えてみよう。

p∈Lを、ある構文とする。このとき、以下の１つが成立するならば、$p \leftarrow q$が成立する。

1. p＝□かつq＝a（$X_1, X_2, \cdots, X_n$）である。
   ただし、Lのn引数の述語記号aに対し、n≧０であり、$X_1, X_2, \cdots, X_n$は、n個の独立変数である。
2. あるアトムPに対しp＝Pかつq＝P｛U／V｝である。ただし、UとVは、Pの中に存在する独立変数である。
3. あるアトムPに対しp＝Pでありn≧０となる、n引数の関数記号Fに対し、q＝P｛V／F（$X_1, X_2, \cdots, X_n$）｝である。
   ここで、Vは、Pに存在する変数であり、$X_1, X_2, \cdots, X_n$は、Pに存在する独立変数ではない。

以下では、L上の具体的な構造的複雑さに対する尺度である構文のサイズについて、Reynold の定義を採用する：構文Pのサイズ、即ち、size（p）はpの記号オカレンス（句読点シンボルを除く）の数であるが、これは、pに存在する独立変数の数を除く。ρ_1 が、この尺度に関する精密化演算子であることを示す。

定理5．4：　ρ_1 は、L上の精密化演算子であるとき、Lのアトムに対して完全である。

証明：pとqの2つの構文について、もし、pがqインスタンスであればp⊢qである。構文のサイズに関する定義より、もし、$q \in \rho_1$（p）であれば、size（q）＝size（p）＋1である。ここで、ρ_1 は、構文からそれらの精密化への写像である。ρ_1 の定義から、次のことが明らかである。任意のp∈Lに対し、ρ_1（p）の集合は、計算可能であり、有限である。ここで、ρ_1（p）（n）の集合は、任意のn≦oに対し計算可能である。そのため、ρ_1 は、ある精密化演算子である。

［Reynolds 70］の定理4は、任意のアトムq∈Lに対し、□からqへの有限の全ρ_1 チェインが存在することを示している。これは、Lのアトムに対し精密化演算子の完全性を定めている。∎

アトムだけを含む公理系は、相対的につまらない。しかしながら、ρ_1 のわずかな一般化は、ある精密化演算子を結論ずける。この演算子は、前述で与えたいくつかの例を公理化するのに十分余裕がある。上で与えた例には、≦，plus，整数リストに対するappendがあった。

定義5．5：ρとρ'をL上の二つの精密化演算子とする。$\rho^* \subset \rho'^*$ であれば、ρ' は、ρよりも一般的であるという。

ρ_1 に関し、次の一般化を考えてみよう。p∈Lを、ある構文とする。このとき、次の一つが成立すればq∈ρ_2（p）である。

1．　q←ρ_1（p）
2．　いくつかのn引数の述語記号aと項t_1，t_2，…，t_n 及び q＝a（t_1，t_2，…，t_n）←a（X_1，X_2，…，X_n）に対して p＝a（t_1，t_2，…，t_n）である。ここで、X_1，X_2，…，X_n は、独立変数であり、X_i はt_j の中に存在する（1≦j≦n）。

Reynoldsは、形式｛P｝←｛Q｝の構文を、変換と呼んだ。我々は、文脈自由変換で、条件2を満す変換を呼びだしている。

いくつか意味のある述語は、アトムと文脈自由変換に対するatomoic-完全公理化をもっており、モデル推論システムでは、それらを推論するための精密化演算子ρ_2が使用されている。

これらの述語は、次のようなものを含む：順序関係、0と後続関数X'により作成される整数のたし算としては、

```
X≦X
X≦Y' ←X≦Y
plus( 0, X, X)
plus( X', Y, Z')←plus( X, Y, Z)
```

空節リスト[]とリスト構造[X|Y]により作成されるリストの前置、後置、後続関係としては、

```
prefix( [], X)
prefix( [A|X], [A|Y]) ←prefix( X, Y)

sufix(X, X)
sufix(X, [A|Y]) ←sufix(X, Y)

subsequeuce( [], X)
subsequence( [A|X], [A|Y]) ←subsequeuce(X, Y)
subsequence(X, [A|Y]) ←subsequeuce(X, Y)
```

リストでの結合関係については、

```
append( [], X, X)
append( [A|X], Y, [A|Z]) ←append( X, Y, Z)

conc( [], X, X)
conc( [A|X], Y, [A|Z]) ←conc( X, Y, Z)
```

t(X, Y) と、いくつかの定数により作成される2進木に対するsubtree 関係については、subtree(X, X)

```
subtree(X, X)
subtree(X, t(Y, Z))←subtree(X, Y)
subtree(X, t(Y, Z))←subtree(X, Z)
```

定理5．6： ρ_2 は、Lに対する精密化演算子であり、Lのアトムと文脈自由変換に対して、完全である。

証明：p = {P} ←とq = {P} ← {Q} とする、任意の二つの構文pとqに対し、p⊢qと、size (p) <size (q) が成立する。よって、ρ_2 は、構文からそれらの精密化への写像である。任意のp∈Lに対し、集合ρ_2 (p) は、有限である。したがって、ρ_2 (p)(n) は、任意のn≦0に対して計算可能である。そのため、ρ_2 は精密化演算子である。

前の証明は、ρ_1 がLのアトムに対して完全であることを示している。任意の文脈自由変換 {P} ← {Q} に対して、□から {P} ←への有限な全ρ_1 -チェインが存在するが、このチェインは、またρ_2 -チェインでもある。ρ_2 の定義により ({P} ← {Q}) ∈ρ_2 ({P} ←) が成立する。
ゆえに、□から {P} ← {Q} への有限な全ρ_2 -チェインが存在する。このチェインは、Lのアトムと文脈自由変換に対して、ρ_2 の完全性を定めている。 ∎

文脈自由変換に関して、二つの単純な一般化がある；ここでは、例を述して、それを紹介する。また、それらについての完全な精密化演算子の定義は、読者にまかせる。このような精密化演算子は、次の例で示される公理化を推論するために、モデル推論システムにより利用された。最初の一般化では、多重文脈自由のホーン構文がある。この構文で、稠密な半順序をもつ述語を公理化できる。すなわち、この述語は、０と終端点１をもち、ＸとＹの間の点として解釈するm(Ｘ，Ｙ）をもつ。

```
０≦Ｘ
Ｘ≦１
Ｘ≦Ｘ
m(Ｘ，Ｙ）≦Ｘ←Ｙ≦Ｘ
m(Ｘ，Ｙ）≦Ｙ←Ｘ≦Ｙ
m(Ｘ，Ｙ）≦Ｚ←Ｘ≦Ｚ，Ｙ≦Ｚ
```

２進木のisomorphism については、

```
isomorphic（Ｘ，Ｘ）。
isomorphic( t(Ｘ１，Ｙ１），t(Ｘ２，Ｙ２))←
        isomorphic( Ｘ１，Ｘ２），isomorphic( Ｙ１，Ｙ２）
isomorphic( t(Ｘ１，Ｙ１），t(Ｘ２，Ｙ２))←
        isomorphic( Ｘ１，Ｙ２），isomorphic( Ｙ１，Ｘ２）
```

また、trueとfalse をもつ真偽公理の証明については、

```
satis(true)
satis( and( Ｘ，Ｙ))←satis(Ｘ），satis(Ｙ）
satis( or(Ｘ，Ｙ))←satis(Ｘ）
satis( or(Ｘ，Ｙ))←satis(Ｙ）
satis( not( Ｘ))←unsatis(Ｘ）

unsatis(false)
unsatis( or(Ｘ，Ｙ))←unsatis(Ｘ），unsatis(Ｙ）
unsatis( and( Ｘ，Ｙ))←unsatis(Ｘ）
unsatis( and( Ｘ，Ｙ))←unsatis(Ｙ）
unsatis( not( Ｘ))←satis(Ｘ）
```

文脈自由変換の第２の一般化は、補助述語をもつ、項自由変換である。このクラスの範囲内で、かけ算とべき乗算を公理化できる。

```
times(O, X, O)
times(X', Y, Z) ←times(X, Y, X) , plus( Y, X, Z)

exponent( O, X, O)
exponent( X' , O, O')
exponent( X, Y' , Z) ←
                    exponent( X, Y, W) , times(W, X, Z)
```

リスト反転については

```
reverse( [ ], [ ])
reverse( [A | X], Y) ←reverse(X, Z) , conc( Z, A, Y)
```

補集合関係については

```
subset( [ ], [ ])
subset( [A | X], Y) ←subset( X, Y) , member( A, Y)
```

挿入ソートについては

```
sort( [ ], [ ])
sort( [A | X], Y) ←sort( X, Z) , insert( A, Z, Y)

insert( A, [ ], [A])
insert( A, [B | X], [A, B | X]) ←A≦B
insert( A, [B | X], [B, Y]) ←B≦A, insert( A, X, Y)
```

[Angluin & Shapiro 81]では、これらにクラスの公理の表現能力について、十分な特徴づけが報告だろう。

５．２　精密化グラフ

任意の１階述語Ｌに対して、Ｌ上の精密化演算子ρは、精密化グラフG_ρ（ρ^*，Ｅ）を導く。これは、直接的な巡回グラフであるが、頂点は、ρの構文である。また、edge（ｐ，ｑ）∈Ｅが存在するための必要十分条件は、ｑ∈ρ（ｐ），ｐ∈ρ^*となることである。（ｐ，ｑ）∈Ｅが、size（ｐ）＜size（ｑ）を導出するので、このグラフは、巡回的である。ｐとｑをρ^*における二つの構文としよう。ｐからｑまでのG_ρのパスは、（p_i，p_{i+1}）∈Ｅとする（$0 \leqq i < n$）$p = p_0, p_1, \cdots, p_n = q$の有限例である。$G_\rho$のパスが、有限の全$\rho$チェインにである事に注意しよう。$G_\rho$でｐからｑへのパスが存在するとき、ｑは、$G_\rho$でｐから到達可能であるという；もし（ｐ，ｑ）∈Ｅであり、ｑがｐの後続要素であれば、ｐはｑの先行要素である。

以下は、G_ρに対するいくつかの特性である。

１．　構文ｑが構文ｐ∈ρからG_ρで到達可能であるための必要十分条件は、ｐ≦ｑである、その結果ｐ⊢ｑに限る。
２．　空文□は、任意のｐ∈ρ^*から到達可能でない。
３．　すべてのｐ∈ρ^*は、□から到達可能である。
４．　もし、構文ｐ∈ρ^*がＬのあるモデルＭで真であれば、G_ρのすべての後続要素は、Ｍでも真である；もしｐがＭで偽であればG_ρのすべての後続要素は、Ｍで偽である。

Ｌで、任意に与えられたモデルＭに対し、Ｍで真なる≦に関した最小構文の集合を定義する。

定義５．７：　ＭをＬのモデルとし、ρをＬ上の精密化演算子とする。ｐがＭとすべての構文ｑ∈ρ^*で真になれば、構文ｐ∈ρ^*は、ρに関してモデルＭのソース構文（source sentence）と呼ばれる。ここで、ｑ≦ｐはＭで偽である。ρに関するＭのソース構文の集合は、モデルＭのソース集合（source set）と呼ばれ、L_M^ρと示される。

ソース構文の定義により、G_ρで空節□からｐへのすべてのパスでのすべての構文がＭで偽になるならばｐ∈ρ^*は、ソース構文である。

$p_0, p_1, \cdots\cdots$の無限な構文列は、$i > 0$　に対して$p_i \leqq p_{i+1}$（又は、$p_{i+1} \leqq p_i$）が成立すれば、無限の昇順列（又は降順）ρ-チェインと呼ばれる。ｑ∈ρ（ｐ）は、size（ｐ）＜size（ｑ）を導出するので、Ｌで無限な降順ρ-チェインは、存在しない。

ゆえに、すべてのp∈Lに対し、集合{q | q≦p}が有限であり、ソース構文でないMで真なるすべての構文p∈ρ^*に対し、q≦pを満すソース構文qが存在する。あるモデルのソース集合は、次の特性をもつ：

定理５・８：　Lを１階述語の言語とし、MをLのモデルとする。ただし□∈L_0 ⊂L_h ⊂Lが成立する。また、ρをL_hに完全なL上の精密化演算子とする。もし、L_h（k_0）が、あるk_0≧0対し、Mの有限なL_0-完全公理を含むとするならば、L_h^n（k_0）は、また、そのような公理化である。

証明：T⊂L_hをMの有限なL_0-完全公理化であるとし、k_0をTの任意の構文の最大数とする。もし、TがL_h^n（k_0）の部分集合でないとすると、ソース構文でない構文p∈Tが存在する。モデルに関するソース構文の定義によりq≦pとなるソース構文q∈L_h^n（k_0）が存在する。

Tのpをqで置き換えてみよう。q≦pであるので、q⊢pが結論づけられる。qが、Mで真で、Tが，MのL_0-完全公理化あるので、構文の集合は、MのL_0-完全公理化でもある。公理化T'のすべての構文がソース構文になるまで、置換のステップを繰り返す。T'は、Mに関する有限のL_0-完全公理化であり、L_h^n（k_0）の部分集合である。L_h^n（k_0）は有限であり、Mで真なので、L_h^n（k_0）は、またMの有限なL_0-完全公理化である。■

もし、T_1とT_2がT_2≦T_1であるような構文の有限集合であれば、T_1 ⊢$_j$ T_2となるような、ある j が存在する。これにより、MがLのh-easyモデルであるならば、Mのソース集合は、Mに関する（h'(n)＝h(n)＋jに対して）h'-easyで有限なL_0-完全公理化を含む。しかしながら、以下の帰納推論アルゴリズムの正当性に対し、ρは、比較的強い要求を満足しなければならない：

定義５．９：　任意の２組の有限集合T_1．T_2に対し、T_2 ⊢$_n$ pがT_1 ⊢$_n$ pを導出するならば、精密化演算子ρは⊢に関して、保存されるといわれる。ただし、構文T_1とT_2の関係は、T_1≦T_2であり、p∈Lである。

もし、証明手続きが、分解証明［Robinson 65］であり、T_1がT_2を吸収（subsume）すれば、T_2 ⊢$_n$ □は、T_1 ⊢$_n$ □を導出する。また、導出は、反ばくの途中にあるので、任意のp∈Lに対して、T_2 ⊢$_n$ pは、T_1 ⊢$_n$ pを導出する。

本論文に記述されているすべての精密化演算子ρに対し、関係$\lesssim$は、吸収関係の部分集合であるので、それらは分解証明に関してすべて保存的（conservative）である。

系５．１０：　定理５．８と同じ仮定の下で、ρがトに関して保存されていれば、hは、全帰納的であり、$T \subset L_h(k_0)$は、ある$k_0 \geq 0$に対し、Mについて有限なL_0-完全で、h-easyな公理化である。

５．３　最も一般的な精密化演算子

本節では、任意に与えられた１階述語Lに対し、演算子ρ_0を定義し、それがLに対する精密化演算子になることを証明する。即ち$\rho^* = L$である。最も一般的な精密化演算子の存在は、理論的な意味で最も興味がある。実用的に、推論される仮説の集合がもつ構造についての情報は、通常、知られている。そのような場合、このクラスに対して完全で、さほど一般的でない精密化演算子を、いつも選んでいる。これは、興味をもっているモデルについて、比較的効率の良い推論を保証している。

以下では、半順序$\lesssim$が任意の１階述語に対して定義されている。Lは、最小要素として□をもつ。精密化演算子ρ_0が定義され、それを計算するためのアルゴリズムのスケッチが与えられる。定理５．１４は、ρ_0がLについて完全であることを示している。また、それは、本節の主な結論である。この結論をのべるために、いくつかのローカルな道具を開発し、$\lesssim$が$\leq$に等しいことを証明する（定理５．１７）。

いくつかの定義から始めよう。$A_1\,\theta < A_2$かつ$B_1\,\theta < B_2$が成立するか、要約して$(A_1 \leftarrow B_1)\,\theta < A_2 \leftarrow B_2$が成立すれば構文$p = A_1 \leftarrow B_1$が、置換$\theta$に対して構文$q = A_2 \leftarrow B_2$を包含する。pがqを吸収し、qがpを吸収するときp≅qと書く。関係≅は、等価関係である；その下にpの等価クラスを示すために［p］を使う。

包含関係は、空節をもったLの等価クラスの集合に半順序を導く。空節□は、最小要素である。また、もし、pがqを吸収するならば、p⊢qである。

例えば、構文{plus(X, Y', Z')} ←と
{plus(X, Y', Z'), plus(X, Y', W) } ←は等価である；
それらは、共に{plus(X, Y', Z')} ←{plus(Z, Y, Z')} と
{plus(X, O', X')} ←を吸収する。
また、これらは、{plus(X, Y, Z) } ←により吸収される。
size(p) ≦size(q) を導出しないならば、pはqを吸収する。
これより、p≦qのような精密化演算子が存在するのは、pがqを吸収するときに限る。 Plotkin [Plotkin 71] は、≦が成立しないという条件下で、無限に完全な降順チェインの例を与えている。この包含関係は、次の集合に拡張される：構文SとTの任意の二つの集合に対し、Sが丁を包含するためには、すべてのq∈Tに対し、pがqを保存するようなp∈Sが存在することである。

以下の定義と結果は、 Plotkinによる [Plotkin 70,Plotkin 71b, Plotkin 71 a]。S⊂S'とS≅S'がS=S'を導出すれば、アトムの集合Sは、縮小されるといわれる。

いいかえると、Sがそれ自身固有の部分集合に等しくないならば、Sは、縮小される。もし、AとBが縮小されれば、構文p=A←Bは、縮小される。構文は反転するのでもしpとqが縮小されp≅qであれば、pとqは変数の名前をつけかえることに等しい。

Plotkinは、与えられた構文pがp≅qを満たす縮小された構文qを計算するアルゴリズムを示している。このアルゴリズムは、それらの等価性に対する構文をしらべるために使用され得る。本節では、‘構文’は、‘この構文に関する等価クラスの縮小代表’を意味する。

定義５．１１：　置換θがアトムの集合を縮小するというのは、
|Sθ|<|S|を満足するときである。置き換えが構文p=A←Bを縮小するというのは、A又はBのどちらか縮小しているときである。

定義５．１２：　pとqがLの2つの構文であるとする。このとき、pを縮小しない置換θが存在し、pθ⊂qが成立するときp$\leqq_0$ qと書く。

≦の定義により、p≦qは、pがqを包含するが、その逆は真でない。

空の置換に対しては、p≦pでありp≦qかつq≦pは、pが変数名の付け換えに匹敵してqに等しい。関係≦は、推移的である。

それは、非縮小の置換にθ_1に対しp$_1$ θ_1 ⊂P$_2$ および非縮小の置換θ_2に対するp$_2$ θ_2 ⊂p$_2$が、次のことを導出するからである。

$\theta_1 \circ \theta_2$は、p$_1$とp$_1$ $\theta_1 \circ \theta_2$ ⊂p$_3$に対して非縮小の置換である。

また、空節の置き換えに対するすべてのp∈Lに対して□≦pである事に注意しよう。そのため、≦は、Lの構文（等価クラス）上の半順序であり、Lは、最小要素として□をもっている。

上の定義から、二つの縮小構文p，qが、size（p）<size（q）導出していることが明確になる。

ここで、PとQをLのアトムとし、Uをアトム又は構文の集合とする。PがUに関して、Qよりも一般的であるとは、Pθ＝Qかつ Uθ＝Uが成立する置換θが存在することである。A←Bを、縮小構文とする。A←Bに完全してPよりも一般的であるような任意のQに対して、構文A←B∪｛Q｝が縮小されなければ、A←B∪｛P｝は縮小される。このときPは、最も一般的であるという。同じことが、A∪｛P｝←Bに対してもいえる。

定義５．１３：　p＝A←Bを、Lの縮小構文とする。このとき、以下の１つが正しければ、q∈ρ_0（p）である。

1．　q＝pθ、ここでθ＝｛V／W｝は、pを縮退しない。また、VとWは、pの独立変数である。

2．　q＝pθ、ここで、θ＝｛V／f（X_1，…，X_n）｝は、pを縮退しない。fはn引数の関数であり、Vは、pの独立変数である。
X_i、1≦i≦nは、pに存在しない独立変数である。

3．　q＝A∪｛P｝←B、ここでPはA←Bに関する最も一般的なアトムである。A←Bに対して、A∪｛P｝←Bは縮小される。

4．　q＝A←B∪｛P｝、　は、A←Bに関して最も一般的なアトムである。A←Bに対してA←B∪｛P｝は、縮小される。

定理５．１４：　ρ_0 は、Ｌ上で完全な精密化演算子である。

証明：$q = \rho_0(p)$ であれば、ｐはｑを吸収する。
そのため、$p \vdash q$ が成立する。１と２の場合での置換は、非縮退的であり、それらは、１つずつｐのサイズを増加している。
３と４の場合、あるアトムが、その残りの部分を変えずにその構文に追加されている。これより、$q \in \rho_0(p)$ は、$size(p) < size(q)$ を導く。
したがって、ρ_0 は、構文からそれらの精密化への写像である。

１と２は、有限に多くの構文ｑにだけ成立するが、一般に、３と４を満足する無限に多くのアトムＰが存在する。以下では、いくつかの定められた範囲よりも小さなサイズのアトムをすべて見つけるために効率的な手続きを記述する。これより$\rho(p)(n)$の集合は、計算可能である。したがって、ρ_0 は、Ｌ上の精密化演算子である。

定理５．１７では、$p \leq_\Delta q$ ならば、ｐからｑへの有限の全ρ_0 -チェインが存在することを示した。任意の構文$q \in L$に対して、$\square \leq_\Delta q$なので、ρ_0 がＬに対して計算可能であることを導く。■

$\rho(p)(n)$ の集合を計算するためのアルゴリズムをスケッチする。
$\rho(p)(n)$ には、構文ｐと$n \geq 0$が与えられている。このとき、ρ_0 が完全であることの証明を続ける。１と２の場合に、非縮退置換の計算は、簡単である。また、有限に多くの構文が、この演算により生成される。３と４の場合、Ａ←Ｂに関する最も一般的なアトムＰを見つけなければならない。Ａ←Ｂは、精密化される構文であり、結果として得られる構文Ａ∪｛Ｐ｝←Ｂ（又はＡ←Ｂ∪｛Ｐ｝）は縮小される。
この集合は、一般に無限である。例として、次の構文を考える。
｛p(X, Z) ｝←｛p(X, f(Y)), p(f(Y)), Z) ｝
アトムに関する以下の無限リストは、
p(f(X), f(Y)), p(f(f(X)), f(Y)),
p(f(X), f(f(Y)), p(f(f(X)), f(f(Y))),…
ρ_0 の定義で、４の場合を満足する。しかしながらsize≦ｋのすべてのアトムは、以下の方法でシステマチックに生成される：
次の条件を満すアトム$p(X_1, X_2, \cdots, X_n)$を選択してみよう。
ｐは、Ｌのｎ引数の述語記号であり、$X_1, X_2, \cdots, X_n$ は、Ａ←Ｂに存在しない独立変数である。

連続的に、Pに存在する独立変数を選択し、以下の演算の１つを実行する：

１．　Pに存在するが、A←Bに存在しない独立変数Uを選択し、
　Pを、P｛V／U｝とする。
２．　n≦0に対するn引数の関数記号fを選択し、
　PをP｛V／f（Z_1，Z_2，…，Z_n）｝とする。
　ここで、Z_i（1≦i≦n）は、P又はA←Bに存在しない変数である。
３．　A←Bに存在する変数Uを選択し、PをP｛V／U｝とする。

A∪｛P｝←Bが縮小されるかsize(P) ＞kになるまで、行なわれる。
もし、２番目の条件が成立すれば、failする。
そうでなければ、構文A∪｛Q｝←Bが任意のアトムQに対して縮小されないことが示される。このアトムQは、A←Bを変化させない置換に対してQ→Pを成立させている。もし、以上が示されればPを返す。

アトムPに演算１及び２を適用したそれぞれの結果は、P→P' でありその場合、size(P')＝size(P) ＋１である。
３番目の演算は、A←Bの変数内で、有限に多くの時間を要す。
これより、このアルゴリズムは、すべての計算パス上で終了される。
演算１と２はLのすべてのアトムを生成するのに十分である。これは、変数名の付け換えに等しい。以上から、この手続きによりsize≦nのすべてのアトムが生成されることが保証されている。演算３は、Pで変数の付け換えを適用するためのメカニズムを与えている。そのため、このアルゴリズムの連続計算により生成されるアトムの集合は、次のように要求された集合である。
以下の２つの補題と１つの定理は、ρ_0 がLに対して完全であることを示している。

補題５．１５：　pとqをいくつかの置換に対して、pθ＝qである二つの構文とする。ただし、θは、pを縮小しない。このとき、pからqへの有限な全ρ_0 チェインが存在する。

証明：この補題は、Reynold の論文における定理４の一般化である。この定理の証明の吟味は、有限の全チェインを超えるために Pと Pθに、その定理が適用されることで示される。 ∎

補題５．１６：　$p = A_1 \leftarrow B_1$ と $q = A_2 \leftarrow B_2$ を $p \subset q$ となる２つの縮小構文とする。このとき、pからqへの有限の全 ρ_0 -チェインが存在する。

証明：この証明は、nの帰納による差集合 $A_2 - A_1$ 及び $B_2 - B_1$ のアトムの総和である。もし、$n = 0$ ならば、$p = q$ であり、空節チェインは、補題を満足する。この補題は、いくつかの $n \leqq 0$ に対して、真であると仮定している。また、$A_2 - A_1$ と $B_2 - B_1$ に $n+1$ 個のアトムが存在している。

Pを $A_2 - A_1$ のアトムとし、A'_2 を $A_2 - \{P\}$ とする（対称的引数は、$A_2 = A_1$ の場合、$P \in (B_2 - B_1)$ に適用する）。帰納的仮定により、
pから $A'_2 \leftarrow B_2$ までの全 ρ_0 -チェイン

$$p = p_0, p_1, \cdots, p_j = A_2 - \{P\} \leftarrow B_2$$

が、存在する。

もし、P が $A'_2 \leftarrow B_2$ に関して、最も一般的なアトムであれば、即ち、$A'_2 \cup \{P\} \leftarrow B_2$ が縮小すれば、ρ_0 の定義により、$(A_2 \leftarrow B_2) \leftarrow \rho_0 (A'_2 \leftarrow B_2)$ であり、この補題が証明される。
それ以外は、$A'_2 \leftarrow B$ に関してPよりも一般的であるようなアトムQが存在する。また、$A'_2 \cup \{Q\} \leftarrow B_2$ は、縮小される。P' を最も一般手なアトムとする。ρ_0 の定義により、$(A'_2 \cup \{P'\}) \in \rho_0 (A'_2 - B_2)$ が成立する。
P' の選択により

$(A'_2 \cup \{P'\} \leftarrow B2)\theta = (A'_2 \cup \{P\} \leftarrow B_2)\theta = A_2 \leftarrow B_2$ が

成立するような置換 θ が存在する。これは、θ が $A'_2 \cup \{P'\} \leftarrow B_2$ を縮退しないことを示している。これより、補題５．１５より有限の全 ρ_0 -チェイン $p_{j+1}, p_{j+2}, \cdots, p_k$ が存在する。
故に、$p_0, p_1, \cdots, p_j, p_{j+1}, \cdots, p_k$ は、補題を満足する。∎

定理５．１７：　$q \lesssim p$が成立する為の必要十分条件は、$p \lesssim_{\rho_0} q$である。

証明：もし、$p \lesssim_{\rho_0} q$ならば、$0 \leq j \leq n$に対して、$p_{i+1} \in \rho_0(p_i)$が成立するような有限のρ_0-チェイン$p = p_0, p_1, \cdots, p_n = q$が存在する。$\rho_0$の定義により、すべての$i$（$0 \leq i < n$）に対し、$p_i \theta \subset p_{i+1}$のような非縮退の置換（あるいは、空節）が存在する。
これにより、$0 \leq i < n$に対して、$p_i \lesssim p_{i+1}$が成立し、$\lesssim$の推移性により$p \lesssim q$を導く。
もし、$p \lesssim q$のとき、$p\theta \subset q$を満す、置換θが存在する。
補題５．１５により、有限の全ρ_0チェイン$p = p_0, p_1, \cdots, p_k = p\theta$が存在する。したがって、$p_0, p_1, \cdots, p_k, p_{k+1}, \cdots, p_n$は、$p$から$q$への有限の全$\rho_0$チェインであり、$p \lesssim q$が成立する。■

これは、ρ_0が最も一般的な精密化演算子であり、１階述語Ｌに対して完全であることを示す。

６．一般的な増殖的モデル推論アルゴリズム

枚挙的アルゴリズム１に使用されている方法は、最も簡単な説明的推測を選択し、いくつかの事実と一致しないことが発見されるまでそれを保持するためのPopperのアイデアに関するだまされやすいインプリメンテーションとして見ることができる。

これが生ずると、この推測に対するある修正を捜さなければならない。即ち、この事実と一致する次の比較的簡単な推測を、選択する。

しかしながら、アルゴリズム１に存在するこのアイデアのインプリメンテーションは、たしかにもっともらしくない。さらに良いインプリメンテーションを実行するためには、古い推測を捨て、取り消しから新しい推測を探す代わりに、仮説の古い集合に対する局部的修正を行い、新しい事実に対抗する仮説を加えることが必要である。

この節では、矛盾探索アルゴリズムと精密化演算子の利用方法を記述する。これにより、新しい事実と一致する推測を局所的に修正し、増殖的方法で、せまい範囲で進化するモデル推論アルゴリズムを得る事ができる。

Figure 6-1は、そのようなアルゴリズムをスケッチしている。

Figure 6-1：増殖的モデル推論アルゴリズムの概要

```
Tに{□}を、設定する。
repeat.
    次の事実を読む。
    repeat.
        while 推測Tが強すぎる do.
            矛盾探索アルゴリズムを適用し、
          Tから反ばく仮説を除去する。
        while 推測Tが弱すぎる do.
            以前に反ばくされた仮説に対する、精密な仮説を、
          Tに付け加える。
    until 推測Tが強すぎることもなく弱すぎることもない。
          （事実の読み込みが続く限り関係する）
forever.
```

推測が、強すぎるか弱すぎるかどうかという問題は、一般的に非決定的である。また、我々は、いくつかの有限資源をもつ計算により、それを近似することができる。

導出の複雑さという制限の下で、いくつかの固定的な帰納関数を混合すると、任意のh-easyモデルの極限で、推論可能なアルゴリズムを生ずる。この概要に基づくアルゴリズムは、充足的である（定義３．３を見よ）ので、定理３．４は、そのようなアルゴリズムがいくつかの固定的な帰納関数ｈに関するh-easyモデルだけの極限で推論できると保証している。従って、それは、この集合の最もパワフルなアルゴリズムである。

このアルゴリズムは、他の問題を提出している：理論が弱すぎるとき、どこに精密化が付加されるべきか？　このアルゴリズムがいくつか自然な方法で徹底されていれば、このアイデアがインプリメントされる正確な方法は、このアルゴリズムの正当性に対する精神論である。

実用的に、この問題は、重大であるが、この推測に精密化の追加を順序ずける良いヒューリスティクスは、このアルゴリズムの収束をかなり高速化くすることになるかも知れない。

本節の残りで、このアルゴリズムのさらに詳細な説明を行う。また、このアルゴリズムが、任意のh-easyモデルの極限で推論されることを示す。いくつかのインプリメンテーション上の論点は、以下で示す。

６．１　あるモデルに関するソース集合の近似

系５．１０では、あるモデルのL_0-完全公理化を探すための正しい立場は、ソース集合の中に存在すると述べている。あるモデルのソース集合は、一般に無限である。我々は、それを近似する方法を、任意に要求される程度で示す。

G_ρ（L_h，E）を、いくつかの精密化演算子ρの精密化グラフとする。ρは、L_hに関して、完全である。G_ρのマークmは、L_hの構文のある集合である。L_hは、‘false’とマークされようといている。G_ρのマークmが、無矛盾マークであるためには、$p \leq_\rho q$の関係がある‘false’とマークされた任意のqと、任意の$p \in L_h$に対しPは、また‘false’とマークされていることである。それが無矛盾なマーク付けであり、‘false’とマーク付けられているすべての構文が、Mで偽になれば、マーク付けmはMに無矛盾である。

定義６．１：　mをG_ρに関し、無矛盾なマーク付けであるとする。構文$p \in \rho^*$がマーク付けmのソース構文と呼ばれるためには、pが‘false’とマーク付けされなくかつ$q \leq_\rho p$となるすべての構文$q \in \rho^*$がmで‘false’とマークされていることである。マーク付けmのソース構文の集合は、マーク付けmのソース集合と呼ばれる。また、この集合は、L_m^sと示される。

もし、あるマーク付けmが、モデルMで無矛盾であれば、そのとき、そのマーク付けのソース集合L_m^sは、L_m^sの近似である。L_m^sは、そのモデルのソース集合である。近似の概念の鍵を定める前に、mがある無矛盾なマーク付けであるとすると、集合L_m^s（k）が任意のk≧０に対し、計算可能になることを示す。これを調べるために、いくつかの構文$p \in L$を考えよう。
もし、pが‘false’とマーク付けされているかsize（p）＞kであれば、$p \in L_m^s$（k）である。それ以外は、（q，p）$\in$Eとなるすべての構文qを考えよ。それらに関して、有限に多くの構文が存在する。もし、それらのすべてが‘false’とマーク付けされているならば、$p \in L_m^s$である。それ以外は、$p \in L_m^s$である。

補題６．２：　ρをL_hに対し、完全なL上の精密化演算子であるとする。G_ρ（L_h，E）をρの精密化グラフとする。Mを、Lのモデルとする。mを、Mで無矛盾なG_ρのマーク付けであるとする。このとき、任意のk＞０に対し、$L_\rho^m(k) \leqq L_\rho^n(k)$が成立する。さらに、Mで偽なる$L_h$（k）のすべての構文が‘false’とマーク付けされているならば、$L_\rho^m(k) = L_\rho^n(k)$である。もし、$L_h$に関するすべての偽の構文が、‘false’とマーク付けされているとき、$L_\rho^m = L_\rho^n$である。

証明：この補題の仮定の下で、Mで真なる任意の構文$p \in L_h$に対して$L_\rho^m \leqq \rho\{p\}$が成立する。これは、L_ρ^mの定義による。特に、いくつかのk≧０に対しsize(p) ≦kが成立すれば、そのとき、$L_\rho^m(k) \leqq \rho\{p\}$が成立する。$L_\rho^m$は、Mで真であるので、$L_\rho^m(k) \leqq L_\rho^n(k)$が成立する。

さらに、Mで偽なるすべての構文L_h（　k）は、‘false’とマーク付けされていると仮定する。L_ρ^mの定義により、$L_\rho^m(k)$は、Mで真である。Mで真である任意のPに対し$L_\rho^m \leqq \rho\{p\}$が成立するので、$L_\rho^n(k) \leqq L_\rho^m(k)$が成立する。$L_\rho^m(k) \leqq L_\rho^n(k)$という事実と共に、等式$L_\rho^m(k) = L_\rho^n(k)$が成立すると、主張する。明らかに、すべてのk≧０に対し、$L_\rho^m(k) = L_\rho^n(k)$であれば、$L_\rho^m = L_\rho^n$が、成立する。

主張の証明：pを$L_\rho^n(k)$の構文とする。$L_\rho^m(k) \leqq L_\rho^n(k)$であるので、$q \leqq p$となるいくつかの$q \in L_\rho^m(k)$が存在する。$L_\rho^n(k) \leqq L_\rho^m(k)$であるので、$p' \leqq q$なるいくつかの$p' \in L_\rho^m(k)$が存在する。
≦の推移性より、$p' \leqq p$が成立する。あるモデルのソース集合の定義より、$p' \leqq p$はあり得ない。故に$p' = p$である。ただし、pは、p＝qを導出する。この対照的議論を、$L_\rho^m(k)$の構文pに適用すると、またそれらもまた、この主張を証明する。 ∎

６．２　増殖的なアルゴリズム

これらの概念を用いて、このアルゴリズムを述べてみよう。
アルゴリズム３は、＜L_0，L_h＞が言語に許される組みであると仮定している。
ρは、L_hに対して完全な精密化演算子であり分解証明を保存する。
α_1，α_2，α_3，……は、L_0のすべての構文の固定的に効果的な枚挙である。
F_1，F_2，F_3，……は、Lにおけるいくつかのモデルの枚挙である。
hは、全帰納関数である。

我々は、これらの過程のもとに、以下の定理が成立することを証明する：

定理６．３：　もし、ＭがＬに対するh-easyモデルならば、アルゴリズム３は、極限でＭと一致する。

モデル推論アルゴリズムが、極限であるモデルに一致すると言われるためには、いくつかのｎ＞ｏに対して事実F_nを読んだ後、ＭのL_0-完全公理化を出力し、異なる推測を２度と再び出力しないことである。
我々は、アルゴリズム３が、以下のことを示すことによって、その極限で一致することを証明する。もし、$L_D(k)$がＭに関し、有限なL_0-完全で、h-easy公理化である最小のｋとしてのk_0をもつならば、アルゴリズム３は、実際に、ｋをk_0に至るまで増加するがさらにそれ以上、増加しない。
また、$L_D(k_0)$は、Pointwise に$L_D(k_0)$に収束する。そのようにするために、そのアルゴリズムについて、以下の事実を、定めるが、それはh-easyモデルの枚挙に適用されていると仮定しいいる：

・$L_D(k_0)$がL_0-完全で、かつＭのh-easyモデルであるならば、それは、k_0以上のｋを増やさない。
・もし、$L_D(k_0)$がＭのh-easyなL_0完全公理化となる最小のｋをk_0とすると、少なくともk_0に至るまで、ｋを増加する。
・もし、ｋがk_0まで増やされk_0よりも大きくしないならば、$L_D(k_0)$は、Pointwise に$L_D(k_0)$に収束する。

ともに、これらの事実は、定理６．３を証明する。

証明：マークの数に対する帰納により、その補題を証明する。最初のマーク付けで、□は、‘false’とマークされる。edge（ｑ，□）は、Ｅの中に１つも存在しないので、このマーク付けは、無矛盾である。また、□は、任意のモデルで偽であるので、このマーク付けは、Ｍで無矛盾である。構文が、矛盾探索アルゴリズムによって反ばくされるならば、構文がこのアルゴリズムによって‘false’とマーク付けされる。いくつかのステージで、G_0のマーク付けがＭで無矛盾であり、構文ｐ∈L_Dが矛盾探索アルゴリズムによって反ばくされると仮定する。L_Dの定義により、構文ｐがL_Dで真になるためには、（ｑ，ｐ）∈Ｅとなるすべての構文ｑ∈L_hがまた‘false’とマークされることである。それ故、無矛盾のマーク付けの定義により、‘false’としG_0におけるｐのマーク付けは、無矛盾である。矛盾探索アルゴリズムの正当性により、このマーク付けは、Ｍで無矛盾である。■

アルゴリズム３：増殖的なモデル推論アルゴリズム．

kに0を、S_{false} に{□}を、S_{true}に{}を、セットする。
□を‘false’とマークする。
repeat.
　次の事実$F_n = <\alpha, V>$を読み、αをS_Vに加える。
　repeat.
　　while $\alpha \in S_{false}$ に対し、$L_S^n(k_0) \vdash_n \alpha$である do.
　　矛盾探索アルゴリズムを適用し、反ばく仮説‘false’をマークする。
　　while $\alpha_i \in S_{true}$に対し、$L_S^n(k_0) \vdash_{h(i)} \alpha_i$ である do.
　　kを１増加する。
　until　while ループのどちらにもはいらない。
output　$L_S^n(k_0)$.
forever.

補題６．５：　もし、アルゴリズム３があるモデルMの枚挙に適用され、$L_S^n(k_0)$がMに関し、h-easyなL_0 -完全公理化であれば、このアルゴリズムは、多くてk_0までkを増やす。

証明：補題６．４により、このアルゴリズムにより作成されるグラフG_ρの任意のマーク付けは、Mに無矛盾である。
補題６．２と共に、これは、任意のk≦0に対して、$L_S^n(k_0) \leqq L_S^n(k_0)$が成立することを示している。$\rho$は、導出に関して保存されるので、kを、$k_0$まで増加後、第２の条件のwhile ループは、二度と満すことがない。ゆえに、kは、k_0を超えることはない。■

補題６．６：　h-easyモデルMの枚挙に適用されたアルゴリズム３は、実際にその事実のすべてを読み込む。

証明：Mがあるh-easyモデルとするならば、内部のrepeatループは、すべての新しい事実の読み込みをさえぎっている。
新しい事実F_nがいくつかのn≦0に対して読み込まれると仮定する；
２つのwhile ループが実行される全回数を数え上げている。

Mは、Lのh-easyモデルであるので、いくつかの有限なL_0 -完全で、h-easyな公理化をもつ。Tを、Mの公理化とする、即ち、任意の構文に対して、$T \vdash_{h(i)} \alpha_i$ が成立する。K_0 をTに関する任意の構文の最大サイズとする。
ρは、導出に関して保存されるので、系５．１０より、$L_\rho^n(k_0)$ は、またMのh-easyで、L_0 -完全公理化でもある。
補題５．６より、kは、k_0 を越えて増加しない。すでに、第２のwhile ループは、有限に多くの時間でだけ実行されている。$L_h(k_0)$ は、有限であるので、任意に与えられた時間で$L_\rho^n(k) \subset L_h(k_0)$ が成立し、最初のwhile ループのすべての反復は、少なくとも$L_h(k_0)$ の構文を‘false’とマークする。それは、最初のwhile ループが有限に多くの時間だけで実行され超えることを示す。それ故、内部のrepeatループの反復が終了する。またそれは、有限に多くの反復だけで実行される。∎

補題６．７： いくつかの$k_0 \geq 0$に対して、アルゴリズム３がkをk_0 まで増加し、k_0 を越えないならば、$L_\rho^n(k_0)$ は、Pointwise に、極限で$L_\rho^n(k_0)$ に収束する。

証明：補題６．２により。size≦k_0 の任意の構文に対して、ρがMで偽ならば、実際にそれは、‘false’とマークされる。
もし、$L_\rho^n(k_0)$ が偽の構文（即ち、$L_\rho^n(k_0) \neq L_\rho^n(k_0)$ ）を含めば、実際に矛盾探索アルゴリズムが適用され、いくつかの構文が‘false’とマークされる。
任意のマーク付けmに対し、$L_\rho^n(k_0) \subset L_\rho^m(k_0)$ が成立し、
その最後は、有限であるので、この過程は、有限に多くの時間に生じ得る。
また、$L_h(k_0)$ で、実際にすべて偽の構文が実際マークされる。

kがk_0 まで増加されてから、いくつかのマーク付けmがそのアルゴリズムにより行なわれたとしよう。もし、$L_\rho^n(k_0) = L_\rho^m(k_0)$ ならば、補題が成立する。即ち、背理法によって、$L_\rho^n(k_0)$ は、偽の構文を含み、mは、これ以上変化しないと仮定している。$L_\rho^n(k_0) \subseteq L_\rho^m(k_0)$ かつ$L_\rho^n(k_0)$ はMの
L_0 -完全公理化であるので、$L_\rho^n(k_0) \vdash L_\rho^n$が成立する。ここで、$L_\rho^n$は、Mで真なる観測文の集合である。許容される要求（admissibility requirement ）により、$L_\rho^n(k_0)$ は、いくつかのi≦0に対するその虚偽性$\alpha_i \in L_0$ の証拠をもつ。即ち、いくつかのj＞0に対し、$L_\rho^n(k_0) \vdash_j \alpha_i$ が成立し、α_i はMで偽である。

$n=max\{i, j\}$としよう。このとき、$L_p^n(k_0) \vdash_n \alpha_i$が成立し、n番目の事実$F_n$を読み込んだ後、集合$S_{true}$は、$\alpha_i$を含む。
最初のwhile ループの条件が満足された時点で、矛盾探索アルゴリズムが呼び出され、いくつかの$p \in L_p^n(k_0)$を、反ばくする。このpは、mが変化しないという仮定と矛盾した状態で、'false'とマーク付けされている。
したがって、$L_p^n(k_0)$は、Pointwise に$L_p^n(k_0)$へ収束する。

補題６．８：　$L_p^n(k_0)$がMのh-easyで、L_0 -完全公理化であるとする最小のkをk_0とするならば、アルゴリズム３は、少なくともkをk_0まで増加する。

証明：背理法により、h-easyモデルMの枚挙に適用されているアルゴリズム３はkをk'まで増加し、k'を越えさせていないとする。このk'は$L_p^n(k_0)$がMのh-easyでないとしている。補題６．７により、$L_p^n(k_0)$はPointwise に$L_p^n(k_0)$に収束する。
k'の選択により、$L_p^n(k')$は、Mのh-easyでも、L_0 -完全公理化でもないので$L_p^n(k') \nvdash_{h(j)} \alpha_j$となるいくつかの$j \leqq 0$に対して、$\alpha_j \in L_0$が存在する。nを、最小インデックスとする。
アルゴリズムは、F_iを読み込む前にkをk'まで増加している。また、n_0を$max\{n, j\}$とする。このとき、n_0番目の事実を読み込んだ後、$\alpha_j \in S_{true}$かつk＝k'である。故に、第２番目のwhile ループの条件は、満足され、kは、k'まで増加される。これは、我々の仮定に反する。
これにより、kは少なくともk_0まで増加し、この補題が証明されている。∎

定理６．３の証明：MをLのh-easyモデルとする。k_0を$L_p^n(k_0)$がMのh-easy、L_0 -完全公理化であるような最小の$k \leqq 0$であるとする。
そのようなk_0は、系５．１０により存在する。アルゴリズム３は、Mの枚挙に適用されると仮定する。補題６．５と６．８により。
アルゴリズム３は、kがk_0まで増加させるがk_0を越えさせていない。
補第６．７により、$L_p^n(k_0)$は、Pointwise に$L_p^n(k_0)$に収束する。
$L_p^n(k_0)$は有限であるので、アルゴリズム３がn番目の事実を読み込んでから$L_p^n(k_0) = L_p^n(k_0)$となる$n \leqq 0$が存在する。またmとkは、決して変化しない。ゆえに、n番目の事実を読み込んだ後、アルゴリズム３は、
Mについてh-easyで、L_0 -完全公理化を出力する。また、決して再び異なる推測を出力しない。即ち、アルゴリズム３は、極限でMと一致する。∎

定理６．３を証明することにより、アルゴリズム３は、アルゴリズムと同じぐらい有能であることがわかる。また、アルゴリズムは、充足的なので、
定理３．４はその種の中で最も有能であるといえる。

６．３　インプリメンテーション上の論点

ホーン理論の推論のためのアルゴリズム３の特殊化と、Prologでのインプリメンテーションは、将来の論文で議論する予定である。ここでは、そのような努力の中に含まれる主な論点を、指摘する。

効率の良いアルゴリズムをインプリメントするために取りかからなければならない、１つの論点は、推測に対し、ある仮説を追加又は除去するという結果をいかにすばやく計算するかである。
この問題は、推測（それを‘false’とマークすることにより）から、ある仮説の除去が、S_{true}のある構文を証明不能性にするか否かをテストするための効率的方法を見つけることにある。S_{true}は、以前、その仮説から証明できていた。また、この問題は、その推測（kを増加することによる）に対するある仮説の追加が、S_{false}のある構文を結論づけるかどうかをテストすることにもある。S_{false}は、以前、仮説から証明不能あった。いまや、この複雑な制限で証明可能になっつている。

これらの問題の中で、最初の問題は、仮説と、事実の間の論理的な従属性（logical dependency）を維持することにより、解かれる。即ち、仮説を記録し、事実の導出として利用した（この仕事に対するインプリメンテーション技術の具体的効果については、[Charniak et al.80] を参照せよ）。第２の問題についての一般的に効果的である解を知らないが、この問題は、S_{false}でのすべての構文を二度と証明しないことを避けている。モデル推論システムは、仮説がホーン節である場合に適用されているいくつかのヒューリィスティクスを混合している。

推測における仮説の数を最適化することは、他の問題を解くことにある。
モデル推論アルゴリズムは、推測の中のある仮説の最大のサイズを最小化することを保証している。この推測は、その論理的能力を増加しない多くの不必要な仮説を含むかもしれない。while ループの条件のテストがもつ複雑度が推測中の仮説の数を大きくするので、そのふるまいは、又、アルゴリズムの効率を左右する。

推測を局所的に最適化するための単純なアルゴリズムがある。これは、仮説がホーン節のときである。統計（statistics）が作成されたモデル推論システムのバージョーンでは、十分な最適化をまだ実施していない。

即ち、読者は、それにより作成される最終的な推測の中に不必要な仮説をいくつか見つけることができる。

指摘される他の論点は、Oracle（コーザ）がこのシステムに与える情報の量である。

例えば、推論される述語sort(X, Y) において、変数のタイプが、整数のリストであるとシステムが認識するならば、構文的には正しいが意味的には正しくない公理の生成は、精密化演算子によりさけることができる。

タイプの記述は、このモデル推論システムの中に、まだ組み込んでいない。

モデル推論アルゴリズムの効率を改善する多くのヒューリィスティクスがある。それは、収束のために必要とされる時間と事実の数という意味の効率である。

例えば、仮説言語と観測言語の組立が決定的であれば、即ち、$T \vdash \alpha$であるか否かという問題が任意の$T \subset L_h$と任意の$\alpha \in L_0$に対して決定的であれば、複雑度の制限hは、無視される。この場合、L_hは、制限された形式（５節でのべたすべてのクラスのような形式）のホーン節を、含む。また、この証明手続きは、後方チェイン（ Prolog で使われる分解証明についてのある制限された形式）である。即ち、このスタック上でゴールの重複オカレンスに対する簡単なテストが行なわれるだろう。これらに関係するトピックスは、将来の論文でもっと十分に議論するつもりである。

７．結論

本論文は、事実から理論を推論するための一般的で増殖的なアルゴリズムを与えた。理論的な分析は、複雑な理論的アプローチから帰納推論に至るまでに知られた最も有能なアルゴリズムのいくつかに同等であることを示した。そのインプリメンテーションは、帰納推論と例によるプログラム合成に関して、既存のシステムをしのいでいる。これらの勇気づけは、基礎的なモデルの計算として１階述語を使って可能にさせると信じている。

帰納推論の手段としてロジックが成功したいくつかの理由がある：

・ロジックは、自然な意味をもつ。もし、チューリング機械が、ある入力で、正しくない結果を計算すれば、その有限制御の中で、ある推移が無意味な“誤り”になっている。“誤り”の推移を示す候補に対し、この推移を変えずに、そのチューリング機械をいつも直すことができる。これにより、その機械は、この入力の上で、正しくふるまうようになるだろう。

一方、もし、論理的な公理の集合が偽の結論をもつならば、少なくとも、その公理の一つは、自然な意味で、完全に偽である。この事実は、矛盾探索アルゴリズムのようなエラー検出アルゴリズムの存在を示している。

・ロジックは、その構文と意味の間に親密な関係をもつ。例えば、変数をある項で置き換えるか、ある構文の２つの変数をユニファイするならば、いつも、論理的に比較的弱い（又は等しい）構文を結論づける。同じことは、アトムを構文の条件又は結論に付け加えることによってもいえる。これは、反ばく仮説の論理的能力（計算能力）を弱くする自然な方法が存在する理由になる。あるいは、別のいいかたをすると、これは、自然でかつ計算可能な精密化演算子が存在する理由である。

・ロジックは、単調でかつモジュラーである。公理を追加又は除去することにより公理体系を変えることは、この公理体系に関する表現（計算）能力に明白な効果をもつ：もし、公理を付け加えるならば、さらに多い帰結をもつ（入出力関係）；もし、公理を除去すれば、より少ない帰結をもつ（入出力関係）。
あるプログラムに対し、そのような構文上の改造は計算による予言可能な効果をもつが、このような実用的プログラミング言語は、多くない。

・プログラミング言語としてのロジックの特徴は、ロジックと制御の分離である。モデル推論システムの効率化の方法は、プログラム［ Kowalski 79b ］の“ロジックの成分”だけを推論し、“制御の成分”を特に記述しないで残すことにあると思える。プログラムのロジック成分は、その記述より以上のものを含む。それは、quicksort のロジックプログラムとinsertion sortのプログラム間の差が示している。ロジックプログラムでの可能な制御は、プログラムの最適化の仕事に似ている。プログラムの最適化と例によるプログラム合成の問題は、それらを同時に解くことを止めているように、それ自身、非常にむずかしい。
例による効率的なプログラム合成の仕事を、次の二つの仕事に分離する事を提案する：例によるプログラムの推論（ときどき、効率が悪い）と、プログラムの最適化。その最適化の仕事が、オブジェクトに関し良く定義されたクラス間の写像を行う事であれば、仕事の分離は、遂行を容易にする。それは、ロジックプログラミングに制御を負わせることにより導びかれる写像をもつ場合である。

・ロジックプログラミング言語として、ロジックの特徴は、プログラムとデータの等価性にある。ロジカルな事実は、直ちに、推論されるロジックプログラムの一部分になり得る。この等価性の意味は、リスプのような他のプログラム言語を記述するのに使われる意味よりも強い。この等価性は、モデル推論アルゴリズム又は、モデル推論システムで、帰納推論のさらに理論的論点にまとをしぼるために使用していない。

しかしながら、ユーザと、より優雅に対話するシステムでは、即ち、McCarthy's の Advice Taker [McCarthy 63] の精神では、そのような特性は、非常に有効であるかもしれない。そのようなシステムは事実による理論の推論だけの指向性ばかりでなく、人間のアドバイスによる対話的な理論の創成、獲得、虫取りの指向性もある。“低レベル”の記述（真のグランドアトム）と、“高レベル”の助言（提案された仮説）を記述する均質な言語は、すべての人間の助言を均質に取り扱うアルゴリズムとシステムを考案する力を与えるだろう* 。

［謝辞］

省略。

Ⅰ．モデル推論システムの効率
第５回目のインプリメントにチャレンジ中。
・コンカレントな推論。
・意味的にまちがった仮説の生成を排除するために、タイプの記述を、精密化演算子で利用する。
・推測を局所的に最適化するアルゴリズムと、仮説の追加と問合せ数の最小化に対するimprovedなヒューリスティクス。

以下、原文３８ページ　～　５０ページまで省略。

・実行例。
・参考文献など。

---

* Truth Maintenance System, Belief System を包含する知識獲得システムとして、まとめあげられる。

Acknowledgements.

The development of the ideas in this paper spans into the past a bit further than NSF grant MCS8002447, which I acknowledge.

While in high school, Moshe Kroy introduced me to Logic and Philosophy of Science. His exciting lectures on Popper's philosophy were a traumatic experience in my intellectual development. Dr. Kroy suggested that if we could test epistemological theories and theories of the mind, then they would become science, rather than metaphysics, and that it might be possible to cary out this task with a computer. This may be the reason for my leaving the homeland, philosophy, for the promised land of computer science.

Years later, by then an undergrad in Tel Aviv University, Amiram Yehudai told me that I did not have to invent Inductive Inference, since such a discipline already existed. He pointed me to the Blums' paper, which Jonathan Stavi helped me to decipher. Prof. Stavi also helped me relate Gold's and Blums' results to inductive inference in logic.

At Yale, I could not have been luckier having Dana Angluin as my advisor. Truly thanking her would make this acknowledgement longer than the paper, so I reserve the honor to for a forthcoming, lengthier document. I will just say that there is nothing more trustable than a proof Dana says is OK. If anything in this paper appears to be to the contrary, it is probably because I rewrote it afterwards.

Drew Mcdermott provided crucial insights into logic hacking, in moments of baffle and bewilderment. Although he prefered my system learning about arches and non-arches, rather than inferring atomic-complete axiomatizations of dense partial orders, he was, nevertheless, encouraging all the way.

Bob Kowalski warned me, just in time, that there will be Sirens in my way, singing "I am a cute, good-looking, interesting, quick-shot, irrelevant theoretical question", and that I should never succumb. So whenever I have imagined hearing one, I plugged my ears and did some more Prolog hacking, until it was over. I have failed only once (twice?), and this is a lesson I will never forget. At least until next time.

Even if I have disturbed him in midst of the most intense hack, Bob Nix was always willing to answer any question, be it on English grammar or combinatorics, or listen to my latest new idea. He read the final draft very carefully, and made sure that there were no errors left.

Ernie Davis and Chris Riesbeck read a previous draft, and provided audible feedback. The fact that even Ernie got lost in Section 6 was a conclusive evidence to me that I should rewrite it. I hope it passes Ernie's Test this time.

Without the outstanding implementation of Prolog we have received from David Warren at Edinburgh, this work would have remained yet another theoretical exercise. The pace with which their system matured into a practical and efficient programming tool can not be attributed to the wonders of Logic alone.

Finally, our Tools group provided a comfortable environment for both hacking and writing, and I am thankful for that. Special thanks to Steve Wood, whom I bugged until Z did (almost) exactly what I wanted, and to John Ellis, who wrote the Scribe interface to the typesetter, and then answered a hundred questions about how to use it.

I. Performance of the Model Inference System

The Model Inference System is implemented in the programming language Prolog [Pereira et al. 78]. It was developed hand in hand with the algorithms, problems in the former motivating improvements in the later. The system has had four stable incarnations so far. The first inferred ground complete axiomatizations of regular sets; the second inferred atomic complete axiomatizations of such sets; and the third inferred atomic complete axiomatizations of a first order language with an arbitrary alphabet, provided the axioms were context-free transformations. The fourth incarnation of the system enabled the inclusion of more general refinement operators, and could infer multiple context-free Horn sentences and term-free transformations with auxiliary predicate. So far it has succeeded in inferring atomic complete axiomatizations of dense partial order with endpoints, binary tree isomorphism, list reversal and list subset, multiplication and exponentiation, and satisfiability of boolean formulas.

A fifth implementation is on its way, featuring the possibility of concurrent inference of arbitrarily many programs; typing specification, to be used by the refinement operators to restrict the generation of semantically wrong hypotheses; an algorithm that locally optimizes the conjecture, and improved heuristics for adding hypotheses and minimizing the number of queries (experiments).

Following are examples of application of the different incarnations of the system, with some relevant statistics. In the first two incarnations the programs were all interpreted under the Edinburgh Prolog-10 interpreter version 1.32. In the third and fourth the programs were compiled using the in-core compiler of the Edinburgh Prolog-10 version 3. The computer is a DECsystem-2060, running Prolog-10 under the compatibility package PA1050. In the following, the running time in CPU seconds measured the time the system took to converge to a correct and sufficient axiomatization of the intended model. The decision to terminate the inference process was made by me, since it is an inherent property of an inductive inference algorithm that *it* can never tell whether it has converged. Note that since this decision was subjective, there may be some errors in the axiomatization below (I am aware of one). The reader is encouraged to find them. *Facts* are the number of facts read therein, and *hypotheses generated* is the number of hypotheses generated by the refinement operator during the inference.

In the earlier versions of the system it would simply go through all possible atoms, in increasing order of complexity, asking the user (or a built-in oracle for the model, when the user was tired) whether they are true in the model. The inefficiency of this approach became more evident as the models got more and more complex, since true atoms are scarce in such models. Therefore in later versions of the system the user is capable of specifying some initial set of facts about the models, hence these implementations were more "fact efficient".

I.1 Inferring Regular Sets.

The first order language for this task consisted of the two predicates in(X) and out(X). The terms of the language were the null string Λ, and strings constructed from a variable or Λ and the successor functions 0(X) and 1(X). For convenience these where simulated by lists of 0's and 1's. The hypotheses language contained Horn clauses, where the only structural restriction on them was that the size of the terms in the condition would not exceed the size of the term in the conclusion. This restriction releases us from a need to impose complexity bound on the size of the proofs. In the following axiomatizations, the number associated with an axiom is its sequential number in the order the axioms were generated by the refinement operator.

Inferring Ground Complete Axiomatizations of Regular Sets.

Strings of even number of 1's and even number of 0's.

157 CPU seconds, 64 facts, 613 hypotheses generated.

(in(Λ) ← true), 3
(out(0) ← true), 9
(out(1) ← true), 23
(out(0X) ← in(X)), 12
(in(00) ← true), 37
(out(0X) ← in(1X)), 14
(in(11) ← true), 76
(out(1X) ← in(X)), 26
(in(00X) ← in(X)), 40
(out(00X) ← out(X)), 52
(out(01X) ← in(X)), 171
(out(01X) ← in(0X)), 173
(out(11X) ← out(X)), 91
(in(01X) ← in(10X)), 191
(out(1X) ← in(0X)), 28
(in(10X) ← in(01X)), 70
(out(10X) ← out(01X)), 289
(in(11X) ← in(X)), 79
(in(010X) ← in(001X)), 219
(in(011X) ← in(0X)), 305

Strings of even parity.

16 CPU seconds, 20 facts, 102 hypotheses generated.

(in(Λ) ← true), 3
(in(0) ← true), 9
(out(1) ← true), 23
(in(0X) ← in(X)), 12
(out(0X) ← out(X)), 20
(out(1X) ← in(X)), 26
(in(1X) ← out(X)), 34

Strings that contain 11 as a substring.

41 CPU seconds, 50 facts, 141 hypotheses generated.

(out(Λ) ← true), 3
(out(1) ← true), 6
(in(11X) ← true), 18
(out(0) ← true), 23
(out(0X) ← out(X)), 27
(in(0X) ← in(X)), 45
(out(10) ← true), 72
(out(10X) ← out(X)), 76
(in(1X) ← in(X)), 19

Strings that contain 11 or 00 as a substring.

25 CPU seconds, 31 facts, 188 hypotheses generated.

(out(Λ) ← true), 3
(out(0) ← true), 6
(in(00X) ← true), 17
(out(1) ← true), 23
(in(11X) ← true), 32
(in(0X) ← in(X)), 19
(out(10) ← true), 67
(in(1X) ← in(X)), 33
(out(10X) ← out(0X)), 79
(out(01X) ← out(1X)), 49

Strings that are of even number of ones or contain 00 as a substring.

64 CPU seconds, 69 facts, 383 hypotheses generated.

(in(Λ) — true), 3
(in(0) — true), 9
(in(00X) — true), 10
(out(1) — true), 23
(in(1X) — out(X)), 34
(in(0X) — in(X)), 12
(out(10) — true), 73
(in(100X) — true), 93
(out(01) — true), 37
(out(01X) — out(1X)), 49
(in(11X) — in(X)), 53
(out(11X) — out(X)), 111
(in(10X) — in(01X)), 98
(out(101X) — out(X)), 175

Inferring Atomic Complete Axiomatizations of Regular Sets.

The inefficiency of inferring ground complete axiomatizations of regular sets was due not so much to the algorithm as to the fact that I was looking at the wrong problem. Logical theories are essentially non-deterministic computational devices. A non-deterministic Turing machine has the privilege of succeeding in a computation only when it is accepting a string. For rejecting a string it merely has to fail to accept it, and there is no need for it to explicitly succeed in a rejecting computation, a requirement that a ground complete theory of a set has to satisfy. This consideration naturally led to the reformulation of the regular set inference problem: find an *atomic complete* axiomatization of the intended model, that is an set of axioms that prove in(s) for every s in the intended regular set, and fail to prove in(s) for strings that are not in that set. The first order language for this task consists of only one predicate in(X), with the same terms as before.

The results of the previous application of the system suggested that regular sets can be axiomatized with transformations only. Therefore the hypothesis language was restricted to such axioms. Applying the system to the inference of atomic-complete axiomatizations of the regular sets as before led to remarkable improvements in the running time of the system, the number of facts needed and the number and complexity of the axioms found.

Strings of even number of 1's and even number of 0's.

28 CPU seconds, 35 facts, 75 hypotheses generated.

(in(Λ) — true), 4
(in(00X) — in(X)), 12
(in(11X) — in(X)), 19
(in(010X) — in(1X)), 41
(in(011X) — in(0X)), 48
(in(100X) — in(1X)), 61
(in(101X) — in(0X)), 67

Strings of even parity.

8 CPU seconds, 17 facts; 29 hypotheses generated.

(in(Λ) — true), 4
(in(0X) — in(X)), 5
(in(11X) — in(X)), 24
(in(10X) — in(1X)), 15

Strings that contain 11 as a substring.

11 CPU seconds, 35 facts, 12 hypotheses generated.

(in(11X) — true), 7
(in(0X) — in(X)), 9
(in(1X) — in(X)), 5

Strings that contain 11 or 00 as a substring.

14 CPU seconds, 52 facts, 12 hypotheses generated.

(in(00X) — true), 6
(in(11X) — true), 11
(in(0X) — in(X)), 5
(in(1X) — in(X)), 9

Strings that are of even number of ones or contain 00 as a substring.

10 CPU seconds, 18 facts, 24 hypotheses generated.

(in(Λ) — true), 4
(in(0X) — in(X)), 5
(in(00X) — true), 6
(in(100X) — true), 16
(in(11X) — in(X)), 19
(in(10X) — in(1X)), 15

I.2 Inferring Atomic Complete Axiomatizations of Simple Models.

In the first two stages of the development of the system described above, the first order language used by the system was fixed. In the third stage the language became part of the input to the system. The input is the function and constant symbols of L, and the predicate $p(X_1,X_2,...,X_1)$ to be inferred. This version used the refinement operator ρ_2 (page 24), which is complete for atoms and context-free transformations. The fourth version incorporated some more general refinement operators, which were complete for multiple context-free Horn sentences and term-free transformations with auxiliary predicate. In case the system inferred an axiomatization that uses an auxiliary predicate (such as times, which uses plus), the predicate to be used was also specified in the input. The statistics below are on the performance of this version. The two numbers following the axioms are their sequential numbers as they were generated by the refinement operator, and the number of refinement operations needed to generate them.

Arithmetic.

Ordering Relation.

le(X,Y) — X is less than or equal to Y.
2 CPU seconds, 10 facts, 14 hypotheses generated.

(le(X,s(Y)) — le(X,Y)), 10, 2
(le(X,X) — true), 1, 1
(le(0,X) — true), 2, 1

Addition.

plus(X,Y,Z) — X plus Y is Z.
5 CPU seconds, 13 facts, 67 hypotheses generated.

(plus(X,X,0) — true), 11, 1
(plus(0,X,X) — true), 14, 1
(plus(s(X),Y,s(Z)) — plus(X,Y,Z)), 52, 3

Multiplication, using addition.

times(X,Y,Z) — X times Y is Z.
20 CPU seconds, 13 facts, 205 hypotheses generated.

```
(times(0,X,0) — true), 12, 1
(times(s(X),Y,Z) — times(X,Y,W),plus(W,Y,Z)), 61, 2
```

Exponentiation, using multiplication.

exp(X,Y,Z) — X to the Y is Z.
28 CPU seconds, 18 facts, 252 hypotheses generated.

```
(exp(X,s(Y),Z) — exp(X,Y,W),times(W,X,Z)), 129, 2
(exp(X,0,s(0)) — true), 78, 2
(exp(0,s(X),0) — true), 148, 2
```

Dense Partial Order with Endpoints.

Partial ordering.

le(X,Y) — X to less than or equal to Y.
10 CPU seconds, 60 facts, 61 hypotheses generated.
Note: m(X,Y) is interpreted as some point between X and Y.

```
(le(X,X) — true), 7, 1
(le(X,m(X,1)) — true), 21, 3
(le(X,1) — true), 5, 1
(le(0,X) — true), 1, 1
(le(m(X,Y),Z) — le(Y,Z),le(X,Z)), 57, 4
(le(m(X,Y),Y) — le(X,Y)), 56, 3
(le(m(X,Y),X) — le(Y,X)), 58, 3
(le(m(X,0),X) — true), 49, 3
```

List Processing Programs.

Member.

member(X,Y) — X is a member of the list Y.
2 CPU seconds, 23 facts, 13 hypotheses generated.

```
(member(X,[Y|Z]) — member(X,Z)), 6, 3
(member(X,[X|Y]) — true), 7, 3
```

Prefix.

prefix(X,Y) — X is a prefix of Y.
4 CPU seconds, 17 facts, 52 hypotheses generated.

```
(prefix([],X) — true), 1, 1
(prefix([X|Y],[X|Z]) — prefix(Y,Z)), 50, 5
```

Suffix.

suffix(X,Y) — X is a suffix of Y.
3 CPU seconds, 17 facts, 26 hypotheses generated.

```
(suffix(X,[Y|Z]) — suffix(X,Z)), 19, 3
(suffix(X,X) — true), 5, 1
(suffix([],X) — true), 1, 1
```

Subsequence.

subsequence(X,Y) — X is a subsequence of Y.
8 CPU seconds, 50 facts, 63 hypotheses generated.

```
(subsequence(X,[Y|Z]) — subsequence(X,Z)), 22, 3
(subsequence(X,[Y|X]) — true), 20, 3
(subsequence([],X) — true), 1, 1
(subsequence([X|Y],[X|Z]) — subsequence(Y,Z)), 51, 5
(subsequence([X],[X|Y]) — true), 45, 5
```

Subset, using member.

subset(X,Y) — X is a subset of Y.
31 CPU seconds, 82 facts, 152 hypotheses generated.

```
(subset(X,[Y|Z]) — subset(X,Z)), 34, 3
(subset(X,[Y|X]) — true), 32, 3
(subset([],X) — true), 1, 1
(subset([X|Y],Z) — subset(Y,Z),member(X,Z)), 113, 3
(subset([X|Y],[X|Z]) — subset(Y,Z)), 103, 5
(subset([X],[X|Y]) — true), 97, 5
```

Append.

append(X,Y,Z) — Z is the result of appending X to Y.
10 CPU seconds, 33 facts, 106 hypotheses generated.

```
(append(X,[],X) — true), 12, 1
(append([],X,X) — true), 11, 1
(append([X|Y],Z,[X|W]) — append(Y,Z,W)), 99, 5
```

Conc.

conc(X,Y,Z) — Z is the result of concatenating the symbol Y to the list X.
7 CPU seconds, 29 facts, 66 hypotheses generated.

```
(conc([],X,[X]):-true), 26, 3
(conc([X|Y],Z,[X|W]):-conc(Y,Z,W),true), 64, 5
```

Reverse, using conc.

reverse(X,Y) — X is the reverse of Y.
6 CPU seconds, 13 facts, 104 hypotheses generated.
Note: The axiomatization of reverse using append (page 4) is not term-free, hence could not be inferred using the refinement operators currently implemented:

```
(reverse([],[]) — true), 6, 1
(reverse([X|Y],Z) — reverse(Y,U),conc(U,X,Z)), 32, 3
```

Last.

last(X,Y) — X is the last element in the list Y.
4 CPU seconds, 12 facts, 18 hypotheses generated.

```
(last(X,[X]) — true), 17, 3
(last(X,[Y|Z]) — last(X,Z)), 8, 3
```

Double every second element.

dbl2nd(X,Y) — Y is the list X, where every other element occurs twice.

64 CPU seconds, 49 facts, 229 hypotheses generated.

```
(dbl2nd([],[]) — true), 6, 1
(dbl2nd([X,Y|Z],[X,Y,Y|W]) — dbl2nd(Z,W)), 222, 11
(dbl2nd([X],[Y]) — true), 97, 5
```

Iota, or tails.

iota(X,Y) — Y is the list of tails of X.
18 CPU seconds, 30 facts, 149 hypotheses generated.

```
(iota([],[]) — true), 6, 1
(iota([X|Y],[[X|Y]|Z]) — iota(Y,Z)), 100, 7
(iota([X|Y],[[X|Z]]) — iota(Y,Z)), 97, 7
```

Pack, or one level flatten, using append.

pack(X,Y) — Y is the list of elements of the lists of X.
9 CPU seconds, 24 facts, 105 hypotheses generated.

```
(pack(X,[]) — true), 3, 1
(pack([X|Y],Z) — pack(Y,W),append(X,W,Z)), 36, 3
```

Oddp.

Oddp(X) — X is of odd length.
3 CPU seconds, 13 facts, 17 hypotheses generated.

```
(oddp([X,Y|Z]) — oddp(Z)), 14, 5
(oddp([X]) — true), 5, 3
```

Pair.

pair(X,Y,Z) — Z is a list of pairs of elements of X and Y.
62 CPU seconds, 61 facts, 226 hypotheses generated.

```
(pair(X,[],X) — true), 16, 1
(pair([X|Y],[Z|W],[p(X,Z)|U]) — pair(Y,W,U)), 211, 9
```

Unlabeled Binary tree predicates

Subtree relation.

subtree(X,Y) — X is a subtree of Y.
9 CPU seconds, 40 facts, 32 hypotheses generated.
Note: t(X,Y) is interpreted as a binary tree whose left subtree is X and whose right subtree is Y.

```
(subtree(X,t(Y,Z)) — subtree(X,Y)), 23, 3
(subtree(X,t(X,Y)) — true), 20, 3
(subtree(X,t(Y,Z)) — subtree(X,Z)), 24, 3
(subtree(X,t(Y,X)) — true), 21, 3
(subtree(X,X) — true), 5, 1
(subtree(0,X) — true), 1, 1
```

Tree isomorphism.

isotree(X,Y) — X is isomorphic to Y.
20 CPU seconds, 110 facts, 117 hypotheses generated.
Note: t(X,Y) is interpreted as a binary tree whose left subtree is X and whose right subtree is Y.

(isotree(X,X) — true), 5, 1
(isotree(t(X,Y),t(Z,W)) — isotree(Y,W),isotree(X,Z)), 108, 6
(isotree(t(X,Y),t(Z,Y)) — isotree(X,Z)), 103, 5
(isotree(t(X,Y),t(Z,W)) — isotree(Y,Z),isotree(X,W)), 41, 6
(isotree(t(X,Y),t(Z,X)) — isotree(Y,Z)), 44, 5
(isotree(t(X,Y),t(Y,Z)) — isotree(X,Z)), 38, 5
(isotree(t(X,Y),t(Y,X)) — true), 35, 5

Satisfiability of Boolean Formulas.

Satisfiability, using unsatisfiability.

satis(X) — X is satisfiable.
10 CPU seconds, 40 facts, 66 hypotheses generated.

(satis(1) — true), 2, 1
(satis(not(X)) — unsatis(X)), 14, 3
(satis(not(0)) — true), 7, 2
(satis(and(X,Y)) — satis(Y),satis(X)), 54, 4
(satis(and(1,X)) — true), 39, 3
(satis(or(X,Y)) — satis(X)), 33, 3
(satis(or(X,Y)) — satis(Y)), 35, 3
(satis(or(X,1)) — true), 28, 3
(satis(or(1,X)) — true), 23, 3

Insatisfiability, using satisfiability.

unsatis(X) — X is unsatisfiable.
10 CPU seconds, 45 facts, 67 hypotheses generated.

(unsatis(0) — true), 1, 1
(unsatis(not(X)) — satis(X)), 14, 3
(unsatis(not(1)) — true), 8, 2
(unsatis(and(X,Y)) — unsatis(X)), 63, 3
(unsatis(and(X,Y)) — unsatis(Y)), 65, 3
(unsatis(and(X,0)) — true), 57, 3
(unsatis(and(0,X)) — true), 52, 3
(unsatis(or(X,Y)) — unsatis(Y),unsatis(X)), 39, 4
(unsatis(or(0,0)) — true), 38, 3

II. Glossary of Symbols.

$\circ$	function composition
$\leftarrow$, $P \leftarrow Q$	implication, P if Q
$\prec$	covering relation
$\subset$	subset or equal
$\subsetneq$	strict subset
$\cup$	set union
$\cap$	set intersection
$\geq$	greater than or equal to
$\leq$, $\not\leq$	{not} less than or equal to
$\&$	and
$\neq$	not equal
$\sim$	not
$\vdash$, $p \vdash q$, $\not\vdash$	p {not} derives q
$\vdash_n$, $\not\vdash_n$, $p \vdash_n q$	p {not} derives q in n derivation steps or less
$\square$	the empty sentence
$\cong$, $\not\cong$	{not} equivalent
α, α_1	observational sentences
θ, θ_1	substitutions
ϵ, $\notin$	set membership
Φ_k	the k^{th} step counting function
Λ	the null string
ρ, ρ_1, ρ_0	refinement operators
$\leq_\rho$, $<_\rho$, $\not\leq_\rho$, $\not<_\rho$	partial order induced by ρ
$\rho(p)$	the refinements of p generated by ρ
$\rho^*(p)$	the transitive closure of p under ρ
ρ^*	the transitive closure of $\square$ under ρ
G_ρ	the refinement graph induced on ρ^* by ρ
$\leq_\omega$, $<_\omega$, $\not\leq_\omega$, $\not<_\omega$	a partial order defined on sentences
L	a first order language
L_h	an hypothesis language
L_o	an observational language
L_o^M	observational sentences true in M
M	a model
L_ρ^M	the source set of a model
m	a marking
L_ρ^m	the source set of a marking
p, p_1, q_1	sentences
T, S, T_1, T_n	sets of sentences
A_1, A_2, B_1, B_2	sets of atoms
P, P_1, Q, Q_1	predicates
t, f, t_1, f_1	terms
X, Y, X_1, Y_1	variables
h, $h(n)$	complexity bound

III. A Useless Lemma.

During the attempts to prove that ρ_0 (defined in section 5.3) is a most general refinement operator, I proved the following lemma. This lemma turned out not to be of any use in the proof. However, since I worked so hard to find its right formulation, and then to prove it, I feel the paper will not be complete without it.

Lemma 7.1: Let $S'=S\cup\{P\}$ be a set of atoms such that S is reduced and $P\notin S$. Then S' is not reduced if and only if exactly one of the following holds:

1. There is a substitution θ and an atom $Q\in S$ such that $Q\theta=P$, $P\theta=P$ and $S'\theta\subset S'-\{Q\}$.

2. There is substitution θ such that: $P\theta\in S$ and $S\theta\subset S$, or, in other words, $S'\theta\subset S$.

Proof: The *if* direction is a straightforward application of the definition of a reduced set: if one of the cases holds for some substitution θ then $S'\theta\subsetneq S'$, and S' is not reduced.

To prove the *only if* direction, assume that $S\cup\{P\}$ is not reduced. Then there there is a substitution θ such that $S'\theta\subsetneq S'$. There are two cases:

Case 1: $P=P\theta^k$ for some $k>0$. Since $S'\theta\subsetneq S'$ implies $S'\theta^k\subsetneq S'$, at least one atom $Q\in S$ is mapped into P by θ^k, that is $Q\theta^k=P$, otherwise $S\theta^k\subsetneq S$, in contradiction to the assumption that S is reduced. The substitution θ^k satisfies the first clause of the lemma.

Case 2: $P\neq P\theta^k$ for all $k>0$. If no $Q\in S$ is mapped into P by θ, that is $Q\theta\neq P$ for all $Q\in S$, then θ satisfies the second clause of the lemma. Otherwise, let $Q_1,Q_2,\ldots,Q_n$ be all the atoms in S mapped into P by some multiple of θ, that is, $Q_i\theta^{k_i}=P$ for some $k_i>0$. For every such atom Q_i, k_i is the only multiple of θ by which it is mapped into P, otherwise there is a k such that $P\theta^k=P$, contrary to the assumption of this case. Let $k=\max\{k_1,k_2,\ldots,k_n\}+1$. Then $P\notin S'\theta^k$, hence $S'\theta^k\subset S$, and the second clause of the lemma holds. ■

References

[Angluin & Hoover 80]
Dana Angluin and Douglas N. Hoover.
Regular Prefix Relations.
Submitted.
1980.

[Angluin & Shapiro 81]
Dana Angluin and Ehud Shapiro.
Structural Complexity vs. Expressive Power of Logic Programs.
In preparation.
1981.

[Angluin 78] Dana Angluin.
On the complexity of minimum inference of regular sets.
Information and Control 39:337-350, 1978.

[Bierman 78] Alan W. Bierman.
The Inference of Regular Lisp Programs from Examples.
IEEE transactions on Systems, Man, and Cybernetics 8, August, 1978.

[Blum & Blum 75]
Lenore Blum and Manuel Blum.
Towards a Mathematical Theory of Inductive Inference.
Information and Control 28, 1975.

[Case & Smith 81]
Case J. and Smith C.
Comparison of Identification Criteria for Mechanized Inductive Inference.
to appear.
1981.

[Charniak et al. 80]
E. Charniak, C. K. Riesbeck, D. V. McDermott.
Artificial Intelligence programming.
Lawrence Earlbaum Associates, Hillsdale, New-Jersy, 1980.

[Duhem 54] Pierre Duhem.
The Aim and Structure of Physical Theory.
Princeton University Press, 1954.
Originally published in French in 1906.

[Gold 65] E. Mark Gold.
Limiting Recursion.
Journal of Symbolic Logic 30, 1965.

[Gold 67] E. M. Gold.
Language identification in the limit.
Information and Control 10:447–474, 1967.

[Gold 78] E. Mark Gold.
Complexity of Automaton Identification From Given Data.
Information and Control 37:302-320, 1978.

[Green et al. 74] Green C. C. et al.
Progress Repri on Program Understanding Systems.
Technical Report Stan-CS-74-444, Computer Science Department, Stanford University, 1974.

[Green 69] C. Cordell Green.
Theorem Proving by Resolution as a Basis for Question Answering.
In B. Meltzer and D. Mitchie, editor, *Machine Intelligence 4*, pages 183-205. Edinburgh University Press, Edinburgh, 1969.

[Harding 76] Sandra G. Harding (ed.).
Can Theories be Refuted? Essays on the Duhem-Quine Thesis.
D. Reidel Publishing Company, 1976.

[Klette and Weihagen 80]
R. Klette and R. Wiehagen.
Research in the Theory of Inductive Inference by GDR Mathematicians - A Survey.
Information Sciences 22:149-169, 1980.

[Kowalski 79a] Robert A. Kowalski.
Logic for Problem Solving.
Elsevier North Holland Inc., 1979.

[Kowalski 79b] Robert A. Kowalski.
Algorithm = Logic + Control.
CACM 22, July, 1979.

[Kuhn 70] Thomas S. Kuhn.
The Structure of Scientific Revolutions.
The University of Chicago, 1970.

[Machtey & Young 78]
Michael Machtey and Paul Young.
An Introduction to the General Theory of Algorithms.
Elsevier North Holland, 1978.

[McCarthy 63] John McCarthy.
Programs With Common Sense.
In Marvin Minsky, editor. *Semantic Information Processing*, chapter 7. The MIT Press, Cambridge, Mass., 1963.

[Mitchell 78] Tom Michael Mitchell.
Version Spaces: An Approach to Concept Learning.
Technical Report STAN-CS-78-711. Stanford Artificial Intelligence Laboratory. December, 1978.

[Paterson & Wegman 76]
Paterson M. S. & Wegman M. N.
Linear unification.
In *8'th Annual ACM symposium on Theory of Computing*, pages 181-186. ACM, 1976.

[Pereira et al. 78] L. Pereira, F. Pereira and D. Warren.
User's Guide to DECsystem-10 PROLOG.
Technical Report 03/13/5570, Labortorio Nacional De Engenharia Civil, Lisbon, September, 1978.
Provisional version.

[Plotkin 70] G. D. Plotkin.
A Note on Inductive Generalization.
In B. Meltzer and D. Mitchie, editor, *Machine Intelligence 5*, pages 153-165. Edinburgh University Press, Edinburgh, 1970.

[Plotkin 71a] Gordon D. Plotkin.
Automatic Methods of Inductive Inference.
PhD Thesis, Edinburgh University, August, 1971.

[Plotkin 71b] G. D. Plotkin.
A Further Note on Inductive Generalization.
In B. Meltzer and D. Mitchie, editor, *Machine Intelligence 6*, pages 101-124. Edinburgh University Press, Edinburgh, 1971.

[Popper 59] Karl R. Popper.
The Logic of Scientific Discovery.
Basic Books, New York, 1959.

[Popper 68] Karl R. Popper.
Conjectures and refutations: The Growth of Scientific Knowledge.
Harper Torch Books, New York, 1968.

[Reynolds 70] J. C. Reynolds.
Transformational Systems and the Algebraic Structure of Atomic Formulas.
In B. Meltzer and D. Mitchie, editor, *Machine Intelligence 5*, pages 135-153. Edinburgh University Press, Edinburgh, 1970.

[Robinson 65] J. A. Robinson.
A Machine Oriented Logic Based on the Resolution Principle.
JACM 12, January, 1965.

[Summers 76] Philip Dale Summers.
Program Construction from Examples.
PhD Thesis, Yale University, 1976.
Computer Science Dept. research report No. 51.

[Summers 77] Phillip D. Summers.
A Methodology for LISP Program Construction from Examples.
JACM 24:161-175, January, 1977.

[Van Emden & Kowalski 76]
M. H. Van Emden and R. A. Kowalski.
The Semantics of Predicate Logic as a Programming Language.
JACM 23:733-742, October, 1976.

[Winston 75] P. H. Winston.
Learning Structural Descriptions from Examples.
In P. H. Winston, editor, *The Psychology of Computer Vision*, . McGraw-Hill, New York, 1975.