ICOT Technical Report: TR-007 MAY 1983

Prologへの否定的知識の導入と プログラム検証

坂井 公，宮地泰造

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Ｐｒｏｌｏｇへの否定的知識の導入とプログラム検証

坂井　公　　宮地　泰造
財団法人　新世代コンピュータ技術開発機構（ICOT）

概要

論理型program 言語Prologの拡張として、否定的情報を書けるようにした Pure Prolog with Negation(PPN) を提案する。プログラムの検証を、「一階述語論理式で書かれた仕様をPPN で証明する」ことと考え、その手順を与える。PPN をprogram として実行する場合の質問回答形式とそれらの解釈を与え、実行algorithm を提案する。さらに直観主義の立場から見たPPN の解釈と実行algorithm を与える．これらの証明手順、実行algorithm はともに既存のProlog処理系を基本としたものであり、簡単に実現できる。

１．　序　論

Robinson [Robinson 65]の導出原理は、Horn clause に限定することによって効率的な実行が可能であり、それを応用したprogram 言語としてProlog［Battani 73, Kowalski 74 ,Warren 77］が知られている。Prologのprogram は、Horn clause の中でもdefinite clause だけに制限されているので、program が内部矛盾を含むことがないという特徴をもつが、一方では否定的情報を書けないという欠点もあわせもっている。現実にimplement されているProlog処理系は pure Prologと異なり、cut のような特殊な述語を持つので、否定も書くことができるが、cut を使って書いた否定はわれわれのもつ直観的な否定概念と異なり「証明できない」という意味である。また、cut は、Prolog処理系の制御方式の知識なしでは把握できない述語であり、論理型言語Prologに手続き型解釈を陽に導入することになり、言語特性を著しく破壊している。同時にPrologの解釈を処理系から独立させることを不可能にしている。しかし、Horn clause は本来否定的情報が書けないものではないので、Prologのprogram として一般のHorn clause を許せば、自然に、否定が書けるようにPrologが拡張される。本稿は第２章でこの拡張を行う。この拡張された言語はPure Prolog with Negation (PPN) と呼ばれる。否定を導入することにより、Prologが持っていた内部矛盾がないという特性が失われるので、PPN の無矛盾性を確認する必要が生じるが、この確認は一般のPrologの処理系で簡単に実現できる。

第３章では、PPN で書かれたprogram の検証について論ずる。program 仕様が一般の１

階述語論理式で与えられているとし、PPN program の検証とは、program をtheorem proverとしてのPPN 処理系に与えることにより「program から仕様を証明する」ことと考えると、検証の手続きも、またProlog処理系の中である程度まで扱える。

PPN も、単なるtheorem proverでなく、program 言語である限り、実行とその結果の解釈が明確に定義されている必要がある。PPN の実行は、Prolog同様、質問に対して回答を出すというmechanism ととらえることができる。第４章では、この実行のalgorithm を与え、質問回答形式とそれらの解釈を与える。

第４章に与えるPPN の実行結果の解釈には、不確定な場合、つまり「いくつかの解の候補が与えられ、その中に解は必ず存在するが、どれが解であるかはわからない」という場合がある。しかし、program 言語として PPNを使用する場合、一般に処理系からの解答は確定的であることが要求される。すなわち、「program の実行は具体的に解を見つけるのでなければ意味がない」という立場であり、このような立場は構成的又は直観的と呼べる。第５章では、解が確定的になるための条件を調べ、その条件を第４章のalgorithm に組み入れた直観主義的実行algorithm を与える。

2. Pure Prolog with Negation (PPN)

Horn clause には次の２つのtypeがある。

(1) 否定リテラルのみからなるもの(「 negative clause」と呼ぶ)

(2) 肯定リテラルを１つだけ含むもの(「 definite clause」と呼ぶ)

例１

¬a(X,Y)	…negative clause
¬a(b)∨¬c(X)	…negative clause
a(X)∨¬b(X,Y)∨¬c(Y)	…definite clause

pure Prolog のprogram はdefinite clause だけからできているが、これを拡張してnegative clause も許すようにする。このPrologを「pure Prolog with negation (PPN) 」と呼ぶ。つまり、PPN program は２つの部分からなり、negative clause だけからなる部分を「否定部」、definite clause だけからなる部分を「肯定部」と呼ぶ。

2．1 PPNの構文

PPN の構文を下記のように定義する。ただし、変数、定数、項、素論理式については既知のものとする。本稿では、変数は英大文字で始まる文字列、定数は（関数記号、述語記号を含めて）英小文字で始まる文字列で表わす。

〈定義１〉　PPN 構文

(1) α1 , …, αn が素論理式であるとき、かつそのときに限り

←α1 , …, αn

は、negative clause である。

(2) α1 , …, αn , βが素論理式であるとき、かつそのときに限り

β←α1 , …, αn

は、definite clause である。

(3) ν1 , …, νn がnegative clause であるとき、かつそのときに限り

ν1 .

.

.

.

νn .

は否定部である。

(4) δ1 , …, δn がdefinite clause であるとき、かつそのときに限り

δ1 .

.

.

.

δn .

は肯定部である。

(5) Γが否定部、Δが肯定部であるとき、かつそのときに限り

［Γ△］

はPPN program である。

上の定義はPrologの構文にあわせたものであり、例１をこの構文で書くと、

```
←a(X,Y)              …negative clause
←a(b),c(X)           …negative clause
a(X)←b(X,Y),c(Y)     …definite clause
```

となる。つまり「←」の左側に肯定リテラル、右側に否定リテラル（の¬を取り去ったもの）がくる。直観的な意味づけとしては、Prolog同様、「←」を「ならば」、「←」の左側のを区切る「，」を「かつ」と解釈してよい。

定義の(1),(2),(3),(4) 中のｎは０であってもよい。ただし(1) の場合ｎ＝０とすると

「←」という矛盾を表わすnegative clause が得られるが、「←」を否定部にもつprogram は 2.2節の意味で矛盾していることになる。

２．２　ＰＰＮの無矛盾性

序論で述べたようにPPN program は内部矛盾を含む可能性があるので、本節では、この無矛盾性の確認を行う手続きについて述べる。

PPN program の内部矛盾とは、否定部で否定されている内容が肯定部から証明されてしまうことである。従って無矛盾性を確認するには、次のようにすればよい。

＜手続き１＞　PPN の無矛盾性

(1) 通常のprolog処理系の肯定部をprogram として与え（肯定部は通常のprologと同じである）、否定部の内容を問い合わせる。

(2) 否定部の各clauseが否定している内容の１つでも証明されればPPN program は矛盾しているといえる。つまり、証明された否定部のclause, あるいはその証明に使われた肯定部のclauseのいずれかが間違っている。

例２

```
←  god(X),mortal(X).        god(jupiter)←.
    「神は死なない」          「jupiter は神である」
                             god(parent(X))←god(X).
                             「神の親は神である」
                             mortal(parent(jupiter)) ←.
                             「jupiter の親は死ぬ」
```

肯定部をprogram として通常のprolog処理系に

```
? - god(X),mortal(X)
```

と問えば X = parent(jupiter)　と答が返るので、このprogram は矛盾している。

３．　ｐｒｏｇｒａｍの検証

program の検証を、「１階の述語論理式で与えられた仕様をprogram から証明する」ことと考え、Prolog処理系による証明の手続きを考える。

３．１　仕様記述の構文

仕様を記述する１階の述語論理式（以下では単に式とよぶ）の構文を次のように定義する。ここでも定数、変数、項、素論理式の定義は既知とする。

<定義２>　式

(1) 素論理式は式である。

(2) α，βが式であれば、

¬α，（α∧β），（α∨β），（α⊃β），（α≡β）

は式である。

(3) αが式、X が変数であれば、

∀X α，　∃X α

は式である。

(4) (1),(2),(3) で定義されるものだけが式である。

普通の論理式では自由変数、束縛変数の区別が存在するが、本稿の仕様記述に用いる式は自由変数を含まないことと約束する。従って変数（本稿では英大文字で始まる文字列）は、全て∀か∃かで束縛されていなければならない。また論理演算子∀、∃、¬、∧、∨、⊃、≡はこの順の優先順位を持ち、同じ演算子では右側が優先されるものとし、括弧を省略することがある。

例３　式の例

1) ∀X （p(X)∧q(X)∨¬p(X)

2) ∀X ∃Y （p(X,Y)∨q(Y))

3) ∃Y ∀X p(X,Y)

4) ¬∀X （p(X)∨q(X))

5) ∃X ¬mortal(X)

３．２　Ｐｕｒｅ　Ｐｒｏｌｏｇの場合

最初にpure Prolog で書かれたprogram の場合について検証手続きを与える。

<手続き２>　Prolog programの検証

(1) 検証したい式（目標と呼ぶ）の否定をとりSkolem標準形にする。

例４．　例３の各式について否定をとりSkolem標準形にすると、

1) ∀X （p(X)∧q(X)∨¬p(X)) …（¬p(a) ∨¬q(a))∧p(a)

2) ∀X ∃Y （p(X,Y)∨q(Y))　…¬p(a,Y) ∧¬q(Y)

3) ∃Y ∀X p(X,Y)　…¬p(a(Y),Y)

4) ¬∀X （p(X)∨q(X))　…p(X)∨q(X)

5) ∃X ¬mortal(X)　　…mortal(X)

(各aはSkolem function)

(2) (1) で作った式がHorn clause 以外のclause（２つ以上の肯定リテラルを含むclause) を含む場合(例えば上の4))、Horn clause の範囲を越えるからあきらめ、本稿の対象とは考えない。

(3) (1) で作った式がHorn clouse だけから成れば、その中のdefinite clause のすべてをassertする(Prolog program の１部として加える) 。
たとえば、上の1)の場合 p(a)←、 5) の場合 mortal(X)←をassertする。

(4) (1) で作った式の中のnegative clause について、それぞれを(3) のassertが加わったprogram に対して質問する。

1)の場合　?- p(a),q(a)

2)の場合　?- p(a,Y) と?- q(Y)

3)の場合　?- p(a(Y),Y)

5)の場合　なし

(5) (4) の質問のどれか１つに対して（たとえば2)の場合、p(a,Y) とq(Y)のどちらか一方に対して）yes という答が返れば目標は証明できる（すなわち検証されたと考える）。１つもyes と返ってこなければ目標は証明できない。

例５　三段論法

目標	program
∀X (r(X) ⊃ p(X))	p(X)←q(X).
	q(X)←r(X).

目標をSkolem標準形にするとr(a) ∧¬p(a).

r(a) ←をassertとし、?- p(a) ときくとyes と答が返る。

例６

目標	program
∃Y ∀X p(X,Y) (例４　3))	p(X,a)←.

?- p(a(Y),Y)と聞くが、X = a(a), Y = aとなりyes と答が返る。

例７

目標	program
∃Y ∀X p(X,Y) (例４　3))	p(X,f(X)) ←.

?- p(a(Y),Y)はoccur check にひっかかり、program とはunify できない。従って答はnoとなる。（この例のようにoccur check は必要である。）

例8

∃X ¬mortal(X) （例4 5）はいかなるprogram に対しても、noという答が返る（質問するgoalがないから）。つまり、目標∃X ¬mortal(X) はいかなるPrologのprogram からも証明されない。

3．3 PPNの場合

＜手続き２＞を PPN用に拡張する。

＜手続き３＞ PPN program の検証

手順：(1),(2),(3),(4),(5) については＜手続き２＞と同じである（もちろんassertも質問も肯定部に対して行われる）。 (5)で１つもyes と答が返ってこなければ(6) へ進む。

(6) 否定部のすべてのclauseについて、(4) と同様の質問をする。質問のどれか１つに対してyes という答が返れば目標は証明できる。１つもyes と返ってこなければ目標は証明できない。

例9

program

否定部	肯定部
←god(X),mortal(X).	god(jupiter)←.
「神は死なない」	「jupiter は神である」

目標

∃X ¬mortal(X) （例4 5）

「死なないものが存在する」

例８でみたようにPrologではこの目標は証明できないが、PPN では否定部があるので、mortal(X) ←を肯定部にassertして、 ?- god(X),mortal(X) と質問すれば X = jupiter すなわちyes と答が返る。従って目標は証明される。

＜注＞ PPN program が 2.2節の意味で矛盾しているなら、任意の目標が証明される。これは、矛盾した論理体系からはいかなる式も証明されることに対応する。

4．ｐｒｏｇｒａｍ言語としてのＰＰＮの実行

まず、実行効率を無視して考える。

pure Prolog への質問（検索又はprogram の実行と考えてもよい）は、negative clauseに対応しているが、PPN はdefinite clause に対応する質問形態も許す。つまり、一つだけ否定リテラルを含む質問を許す。

第３章の検証の場合には、program は目標が証明されるか否かだけを答えればよいが、上の質問に対してはその質問を満足する変数値を答えることが要求される。そこで＜手続き３＞を多少修正した手順を次に与える。

＜手続き４＞　PPN の実行

1) 質問が肯定リテラルだけから成れば、pure Prolog と同様に扱う（この場合、否定的知識は利用する必要がない）.
2) 質問が否定リテラルを含むとする。リテラルの順序を無視すれば、
   ¬α，「　（「は肯定リテラルだけから成る）
   と書ける。そこでα←「を肯定部にassertする。
3) 否定部のclauseの１つを選び、それを 2) のassertが加わったprogram に対して質問する。その際、α←「というclauseが実行されるたびにその各変数にbindされた値を答の候補として覚えておく。もし、実行がsuccess すれば、

```
    変数1 = 値1 , …, 変数n = 値n      ……最初に bind された値
∨   変数1 = 値1', …, 変数n = 値n'     ……２番目に bind された値
                 ・
                 ・
                 ・
∨   変数1 = 値1", …, 変数n = 値n"     ……最後に bind された値
```

   という形式で答える。　α←「を一度も使わずにsuccess した場合、program が矛盾している。

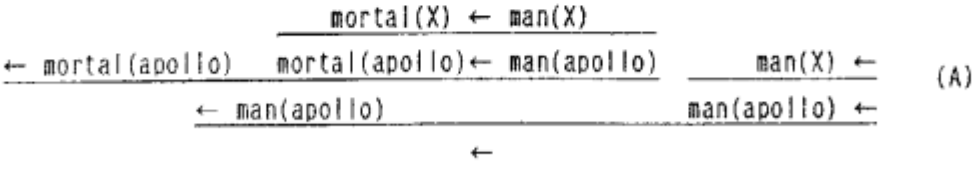
   この答の意味は「変数binding のいずれかの中に与えられた質問を満足するものがあるが、現知識からはどれがそうであるかは決定できない」ということである（例11参照）。
4) 途中でfailしたり、success しても次の答が要求されたりすれば 3) の過程をbacktrack しながら繰返す。もちろん、否定部のclauseの選択もbacktrack の対象となる。

例10

```
program
←mortal(apollo).          mortal(X) ← man(X).
「apolloは死なない」        「人間は死ぬ」
```

質問　?- ¬man(X)
「人間でないものは？」

man(X)← をassertして ?- mortal(apollo)と質問すると



(A)

となり、success する。図の(A) の変数binding を覚えておき、
X = apollo　と答を返す。
更に次の答を要求すれば、back trackを行い、

```
X = parent(apollo),
X = parent(parent(apollo)),
        ・
        ・
```

と順に答が返る。

例11

なぜ＜手続き４＞の 3) で述べたような特殊な意味をもちだす必要があるのか説明する。次のような定理と証明を考えてみよう。

定理：２つの無理数X,Y で X^Y が有理数であるものが存在する。
証明：$\sqrt{2}$ は無理数である。一方、

$$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\times\sqrt{2}} = \sqrt{2}^{2} = 2$$

であるから $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$ は有理数である。
さて、$\sqrt{2}^{\sqrt{2}}$ が無理数であるとすると、
$X = \sqrt{2}^{\sqrt{2}}$、$Y = \sqrt{2}$　とすれば X^Y は有理数である。

$\sqrt{2}^{\sqrt{2}}$が有理数であるとすると、

$X = \sqrt{2}$、$Y = \sqrt{2}$　とすればやはり X^Y は有理数である。

従っていずれにしても定理を満足するX, Yが存在する。□

上の証明に必要な知識は、$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$ が有理数であり、$\sqrt{2}$が無理数であるということだけである。そこで次のようなPPN program を考える。

$\leftarrow ir((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$.　　$ir(\sqrt{2}) \leftarrow$.

「$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$は有理数である」　　「$\sqrt{2}$は有理数である」

質問として上の定理、つまり　?- ¬$ir(X^Y)$, $ir(X)$, $ir(Y)$ を与える。

$ir(X^Y) \leftarrow ir(X), ir(Y)$ を肯定部に追加して、否定部を

?- $ir((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$

と質問すると、

$ir(X^Y) \leftarrow ir(X), ir(Y)$ (A)

$ir((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$　　$ir((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}) \leftarrow ir(\sqrt{2}^{\sqrt{2}}), ir(\sqrt{2})$

$\leftarrow ir(\sqrt{2}^{\sqrt{2}}), ir(\sqrt{2})$

$ir(X^Y) \leftarrow ir(X), ir(Y)$ (B)

$\leftarrow ir(\sqrt{2}^{\sqrt{2}}), ir(\sqrt{2})$　　$ir(\sqrt{2}^{\sqrt{2}}) \leftarrow ir(\sqrt{2}), ir(\sqrt{2})$

$\leftarrow ir(\sqrt{2}), ir(\sqrt{2}), ir(\sqrt{2})$　　$ir(\sqrt{2}) \leftarrow$

$\leftarrow$

上の証明で

(A) で　$X = \sqrt{2}^{\sqrt{2}}$、　$Y = \sqrt{2}$

(B) で　$X = \sqrt{2}$、　$Y = \sqrt{2}$

の変数binding が行なわれているが、これが丁度、前述の証明の２つの場合に対応している。又、明らかに否定部と肯定部の知識だけからは、$\sqrt{2}^{\sqrt{2}}$が無理数か有理数か決定することはできない。従って(A) 、(B) のどちらであるかも決定できない。

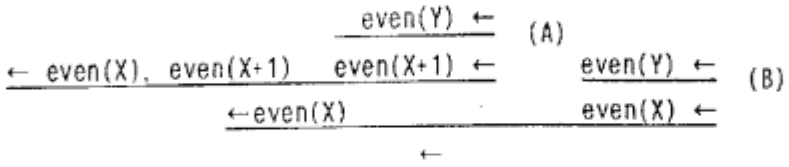
例12　もう１つ例をあげる。

肯定部のない次のprogram を考える。

← even(X), even(X+1).

「X とX+1 がともに偶数ということはない」

?- ¬even(Y)　「偶数でないものは？」と質問すると　even(Y) ←をassertして
?- even(X), even(X+1) ときく。



(A),(B) の変数binding より Y = X+1 ∨ Y = X と答える。つまりX がなんであれ、X かX+1 の一方は偶数でないが、どちらかは決定できないことになる。

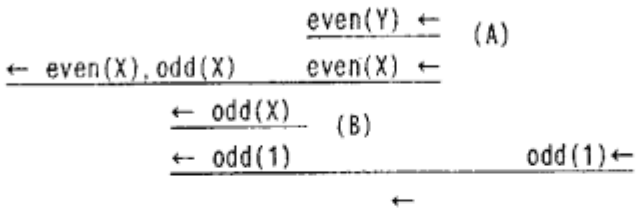
<注>変数binding を覚えておくといっても、clauseが実行されたときのbinding だけをとおりいっぺんに覚えておくのではなく、そのときbindされた値が変数をもち、その変数が後のunification でさらに別の値にbindされたら、それを反映させねばならないのは当然である（次例参照）。

例13　次のprogram を考える。

← even(X), odd(X).　「偶数は奇数でない」

odd(1)←.　「１は奇数である」

?- ¬even(Y)　「偶数でないものは？」と質問すると
even(Y) ←をassertして、?- even(X),odd(X) ときく。



上の証明で(A) でY にX がbindされているが、さらに(B) でそのX に１がbindされているので、その結果を反映して

Y = 1

と答えなければならない。

ところで、＜手続き４＞は通常のprolog処理系にあわせてnegative clause （否定部）から導出を始めるように書かれているが、本来導出原理の適用はどの２つのclauseから始めてもよい。ただprologの場合はnegative clause すなわち質問であったから、negative clause から導出を始めることは当の質問から導出を始めることになり、効率的でもあったわけである。

だが PPNの場合、質問がdefinite clause に対応するもの（否定リテラルを含むもの）の場合もあり、その場合にもnegative clause から導出を始めると、実行効率が低下することは当然予想される。

従って、与えられた質問そのものから導出を開始するalgorithm を考えることが重要である。それを次に与える。

＜手続き５＞　ＰＰＮ　program の実行の効率化

1) 否定を含まない質問についてはPrologと同じである。
2) 否定を含む質問「 ?- ¬α，「」の場合、
   αとunify できるリテラルを否定部の中でさがす。
   ←Β，α'，Δ（α' はαとunify 可能）が見つかれば、そのunify における変数binding を答の候補として覚え、α←「を肯定部にassertする。後は
   ?- Β'，Γ'，Δ' (注) をgoalとして＜手続き４＞と同様である。assertしたα←「がその後使われればその変数binding は答の候補に加わる。
3) 2)が失敗すれば（つまり、αとunify できるリテラルが否定部になかった場合）肯定部のbodyの中でαとunify できるリテラルをさがす。β←Β，α'，Δが見付かれば、そのunify における変数binding の答の候補として覚え、
   ?- ¬β'，Β'，Γ'，Δ' (注) をgoalとして 4) へ進む。
4) β' とunify できるリテラルを否定部の中でさがす。あれば 2) と同様である。ただしβ' に関する変数binding は答の候補に入れない。
5) 4)が失敗すれば、肯定部のbodyの中でβ' とunify 出来るリテラルをさがす。以下3)と同様である。ただしβ' に関する変数binding は答の候補に入れない。
6) 上の過程でgoalがsuccess すれば、＜手続き４＞同様答の候補を出力する。failしたり、次の答を要求されたりすればbacktrack を行なう。

（注）β'，Β'，Γ'，Δ' はαとα' をunify した結果をβ，Β，Γ，Δに反映したものである。厳密に言えば、αとα' のmost general unifierをθとするとき、θ（β），θ（Β），θ（Γ），θ（Δ）である。

5. PPNと直観主義論理

本稿の例11の証明は、排中律を使っており、直観主義論理では証明されないものの例としてしばしば引用されるものである。従って、PPN の処理系は直観主義論理を越えていることになるが、この辺の事情について述べる。

Theorem Proverとしての導出原理はもともと古典論理のための推論規則であり、直観論理とは直接の関連はない。実際、導出原理を適用するためには論理式をSkolem標準形に変換するが、直観論理ではこのような変換は許されない。

しかし、Horn clause に制限されているPrologやPPN の場合には、導出原理は直観主義下でも立派な意味を持つ。その意味とは次のとおりである。

1) negative clause $\leftarrow\alpha_1, \cdots, \alpha_n$ は
$\forall X1 \cdots \forall Xm \neg(\alpha_1 \wedge \cdots \wedge \alpha_n)$

2) definite clause $\beta \leftarrow \alpha_1, \cdots, \alpha_n$ は、
$\forall X1 \cdots \forall Xm (\alpha_1 \wedge \cdots \wedge \alpha_n \supset \beta)$
(X1,…,Xm は$\alpha_1, \cdots, \alpha_n$, β中に現れる全ての変数)

導出原理はこの解釈のもとで直観主義論理体系内でもまったく正当な推論規則である（ただし、古典論理の場合と異なり完全ということはない）。特にPrologのprogram は1)を持たないので 2) の解釈を、さらに、

2') $\forall X1 \cdots \forall Xm (\alpha_1 \supset \cdots \supset \alpha_n \supset \beta)$

に変えることにより、直観主義論理の部分系である positive implicational logic と一致する（ちなみに、2)と 2')は直観主義論理体系の中でも同等である）。

むしろ、導出原理を、Horn clause という枠の中に押込めることによって直観主義的（構成的）なものにし、具体的な解を捜してくる機構と捉えることを可能にしたことが、program 言語としてのPrologの成功であったといえる。さて、例11の証明は、一見 PPNが直観主義論理体系の中に納まらないことを意味しているかのようであるが実はそうではない。PPN が例11で証明したことは、次の３つの式

$\neg ir((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}),\ ir(\sqrt{2}),\ \forall X\ \forall Y\ (ir(X) \wedge ir(Y) \supset ir(X^Y))$

から矛盾が生ずるということである。そしてこれからいえることは、

$\neg ir((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$ と$ir(\sqrt{2})$ とから

3) $\neg \forall X\ \forall Y\ (ir(X) \wedge ir(Y) \supset ir(X^Y))$

が演繹されるということであり、3)を

3') $\exists X\ \exists Y\ \neg ir(X^Y) \wedge ir(X) \wedge ir(Y)$

と解釈されることが許されるのは古典論理の場合である。つまり例11が証明したものは 3) であって 3')でないと考えればPPN は直観主義論理内にとどまっているし、3') だと考えれば直観主義を越えてしまうのである。

しかし、例10のようにassertした質問が証明の際に一度しか使われていない場合は直観主義のもとでも 3')のような解釈が可能であり、その場合は例 11,例12のような不確定的な解釈は不要になる、。このことをより一般的に述べると次のようになる。

＜定理１＞

Hを Horn clause の集合とし、α←Γとから導出原理によって矛盾が導かれるとする。このとき証明中に現れるα←Γへの変数代入について、そのすべての代入結果がunify 可能ならば、直観主義論理において、H（の直観主義的解釈）から、

$\exists X \cdots \exists X_m (\neg\alpha \wedge \Gamma)$　　（X , …, Xmはα, Γ中に現れる全変数）

が証明できる。

＜証明＞

α←Γへの代入結果すべてをunify するunifier をθとする．また証明中に現れるすべての必要な代入をHに行なった結果をH´とすれば、

θ（α）←θ（Γ）とH´から代入なしで導出原理により矛盾を導くことができる。

証明は後述するが、このことから次の1),2) がいえる。

1) 導出原理でH´からθ（Γ）が導かれる。

2) 導出原理でH´とθ（α）から矛盾が導かれる。

導出原理は直観主義でも有効であるから、 2) よりH´から直観主義で¬θ（α）が導かれる。よってH´から¬θ（α）∧θ（Γ）、さらには

$\exists X1 \cdots \exists X_m (\neg\alpha \wedge \Gamma)$

が導かれる。またあきらかにH´はHから導かれるので、Hから

$\exists X1 \cdots \exists X_m (\neg\alpha \wedge \Gamma)$ が導かれる。　□

上述の定理より、＜手続き４＞又は＜手続き５＞を修正して、次のような直観主義におけるPPN の実行algorithm を得られる。

＜手続き６＞　PPN の直観主義的実行

(1) 答の発見に至る過程は＜手続き４＞、＜手続き５＞と同じ。

(2) 答の候補が得られたら、それらがunify できるかどうか調べる。

(3) unify できればそのmost general unifierを答として返す。できなければback trackする。

例14

例10の場合、前述したように、それぞれの答を導くのに、assertしたman(X)←は一度しか使っていない。よって例10の答は直観主義下でも、答としてよい。

例15

例11の $X=\sqrt{2}^{\sqrt{2}}$, $Y=\sqrt{2}$ と $X=\sqrt{2}$, $Y=\sqrt{2}$ は Xにbindされた $\sqrt{2}^{\sqrt{2}}$ と $\sqrt{2}$ が unify できないので答とはみなされない。

例16

例12の $Y=X+1$ と $Y=X$ は、$X+1$ と Xがunify できない（occur check にひっかかる）ので答とは見なされない。

例17

r をRussell の集合 $r=\{X:\neg\ X\in X\}$ としよう。するとわれわれは、次の program のような知識をもっている。

$\leftarrow r\in X,\ X\in X.$

「Russell の集合に含まれるものは
自分自身を含まない。」

これに ?- $\neg\ Y\in Z$ と質問すると、 $Y\in Z\leftarrow$ をassertして、?- $r\in X$ とさく。

$$\dfrac{\dfrac{\leftarrow r\in X,\ X\in X \qquad r\in X\leftarrow}{\leftarrow X\in X}(A) \qquad \dfrac{Y\in Z\leftarrow}{X\in X\leftarrow}(B)}{\leftarrow}$$

(A)の変数binding は $Y=r,\ Z=X$, (B)のbinding は $Y=X,\ Z=X$ であるが、これらはunify でき、結局 $Y=Z=r$ となるので、

$Y=r,\ Z=r$ と答える。

＜定理１＞の証明中で述べた1)と2)の事実を示す。

＜補題１＞

HをHorn clause の集合とする。$\alpha\leftarrow\Gamma$ とHから導出原理により代入なしで矛盾が導けるならば、

1) 導出原理によってHからΓが導かれる。
2) 導出原理によってHとαから矛盾が導かれる。

＜証明＞

2)は明らか。1)を証明する。

Horn clause による導出は次の２タイプしかない。

A)　２つのdefinite clause からの導出（dxd 導出）

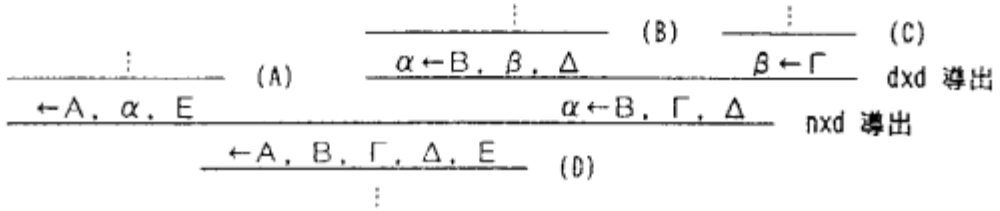
　α←B，β，Δ　と　β←Γ　から　α←B，Γ，Δ　を導出する。

B)　negative clause とdefinite clause からの導出（nxd 導出）

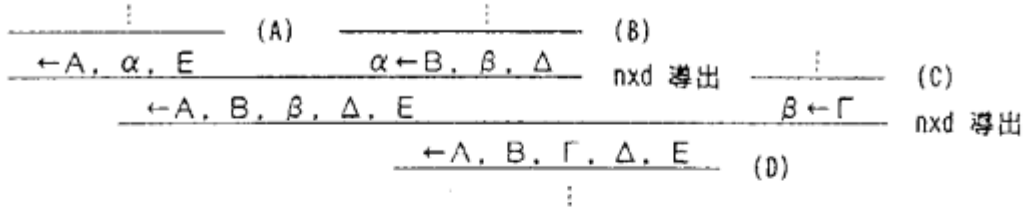
　←B，β，Δ　と　β←Γ　から　←B，Γ，Δ　を導出する。

さて、導出原理を用いたHorn clause からの矛盾の証明は、nxd 導出だけの証明に書き替えることができる。これは次の手続きで行う。

dxd 導出だけでは矛盾を証明することはできないので、証明の中にdxd 導出が存在すれば、必ず次図のようにnxd 導出と接続している部分がある。

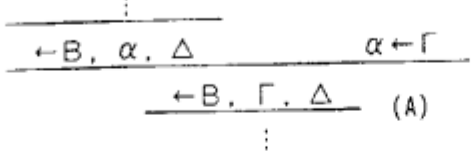


そこを次のように２つのnxd 導出に書きかえれば、dxd 導出の数がへる。



この手続きをdxd 導出がなくなるまで繰り返す。

さて、上の手続きでHとα→Γから矛盾を導く証明をnxd 導出だけのものに書きかえる。すると証明はnegative clause に順次Hのdefinite clause 、もしくは、α←Γが適用された形のものになる。その中で最後にα←Γが適用された部分を、



とする。さて (A)の部分は←B，Γ，ΔとHのdefinite clause だけからの矛盾の証明になっているし、当然代入はないから、次の＜補題２＞によって←B，Γ，Δは、Hのdefinite clause だけから導かれる。よって 1) が成り立つ。□

＜補題２＞

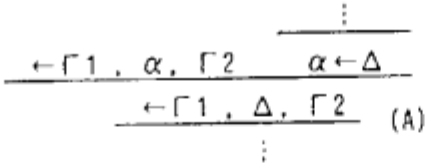
←Γと definite clauseの集合Pから代入なしの導出原理で矛盾が証明されるならば、PからΓが証明される。

＜証明＞

矛盾の証明の段数に関する数学的帰納法による。

1) 段数＝0なら、 ←Γ は ← に他ならない。この場合Γは空であるから、Pから証明すべきものはない。よって、補題は成立する。

2) 段数＝n＞0で、段数がn未満の証明については補題が成立すると仮定する。この場合Γは空でないから最初の ←Γ に対する導出原理の適用箇所は次のような形をしている。



(B)

(Γ＝Γ1 , α, Γ2 である)

(A)

ここで証明の(A) の部分を見ると、この段数は明らかにn未満であり、←Γ1 , Δ, Γ2 とPから矛盾を導く証明になっている。よって帰納法の仮定より、←Γ1 , Δ, Γ2 はPから証明される。

一方、(B) の部分はPからのα←Δの証明である。今，ΔはPから証明されることがわかっているから、αもPから証明可能である。

よって、Γ1 , Δ, Γ2 すなわちΓはPから証明される。□

［謝 辞］

本研究の機会を与えていただいた当（財）新世代コンピュータ技術開発機構の渕一博所長，古川康一室長、有益な意見をいただいた國藤進，向井国昭，黒川利明主任研究員，麻生盛敏研究員に深謝致します。

〔参考文献〕

[Kowalski 74] Kowalski,R.: "Predicate logic as a programming language ," IFIP 74, North-Holland, pp.569-574,1974.

[Battani 73] Battani,G. and Meloni,H.: "Interpreteur du langage de programmation PROLOG, " Groupe d'Intelligence Artificielle,U.E.R. de Luminy, Universite d'Aix-Marseille, 1973.

[Warren 77] Warren,D.H.D.: "Implementing PROLOG — compiling predicate logic programs, " Research Report 39 and 40,Dept.of Artificial Intelligence, University of Edinburgh, 1977.

[Robinson 65] Robinson,J.A.: "A machine oriented logic based on the resolution principle, " J. ACM 12, No 1, pp.23-41, 1965.