Ｐｒｏｌｏｇによるプロダクション・システムに関する一考察

山本　聡、田中　穂積（東京工業大学・工学部）

１．まえがき

ＭＹＣＩＮをはじめとする多くのエキスパート・システムで、プロダクション・システムが使われている。それらの多くは、Lispでインプリメントされているが、近年、それを Prolog でインプリメントする試みがなされている。

プロダクション・システムの推論方式には、前向き推論と後ろ向き推論があるが、後ろ向き推論は、Prologで簡潔に記述できる(1) 。これは、Prologのプログラムのトップダウン実行メカニズムが、基本的に後ろ向き推論と同じ機構であるため、推論の制御を Prolog にまかせる事が出来るからである。一方、前向き推論については、Prologの実行制御機能をそのまま利用できないため、前向き推論を行なうためのルール・インタープリタを、Prolog上に作成する方法がとられている(2),(3) 。

本稿では、まず、プロダクション・ルールをトランスレートする事によって、前向き推論を行なうPrologプログラムを得る方法を提案する。それによれば、後ろ向き推論と同様に、前向き推論を行なう場合もルール・インタープリタを作る必要がない。次に、構文解析システムＢＵＰ(4) の制御構造を応用した、前向き後ろ向き併用プロダクション・システムについて述べる。このシステムにおいても、プロダクション・ルールは、Prologのプログラムにトランスレートされる。

なお、プロダクション・システムを Lisp でインプリメントする場合には、前向き推論、後ろ向き推論共に、ルール・インタープリタを作る必要がある。

２．前向き推論システム

2.1 ルール・インタープリタ方式

Prolog上にプロダクション・システムのルール・インタープリタを作る方式は通常、図１のような形で記述される。すなわち、プロダクション・ルールは、単項節 rule 中に埋め込まれており、このrule節を呼ぶ事により、ルールのひとつを任意に選び出し、test_if で、その前提部を一つ一つチェックしてゆく。

2.2 ルール・トランスレート方式

Prologの利点を生かした前向き推論機構として、我々は、ルールを Prolog プログラムにトランスレートして実行するシステムを提案する。これは、文法規則を Prolog プログラムに変換して実行するＤＣＧと同じような考え方であり、図２のように、ユーザーの書いたプロダクション・ルールは、トランスレータにより Prolog プログラムに変換される。したがってプロダクション・システムのルール・インタープリタを作る必要がない。図３に、その例を示す。図３中の、deduce_1節は、図１の rule 節に書かれたプロダクション・ルールが、変換されたものである。すなわち、

```
a <= b,c,d.
```

の形のプロダクション・ルールは、

```
deduce_1(a,Facts):-
        member(b,Facts),member(c,Facts),
        member(d,Facts),check(a,Facts).
```

の形に変換される。この deduce_1 節を呼ぶ事により、現在わかっている事実 Factsから推論できる事実を導出する。

2.3 考察

ルール・インタープリタ方式では、実行時にインタープリタがルールを読みながら動作する。ルールが複雑になり、また、ルールに notや or などを導入した場合、インタープリタは、実行時にそれらに対する解釈を行なわなければならない。一方、ルール・トランスレート方式では、ルールは Prolog プログラムに変換されるので、それらの制御はPrologの実行機能にまかされる。ルールに notや or を導入した時も、それらの解釈は Prolog に組み込みの実行機構にまかされるため、実行速度が速い。

なお、ここに示したプロダクション・システムは、簡単なものであり、一度証明された事実の再計算を避ける機能がない。この点は今後の重要な課題である。

```
deduce(Facts,Final_Facts):-
        deduce_1(Conseq,Facts),deduce([Conseq|Facts],Final_Facts).
deduce(Final_Facts,Final_Facts).
deduce_1(Conseq,Facts):-
        rule((Conseq <= Premis)),test_if(Premis,Facts),check(Conseq,Facts).
test_if( (F,Fs),Facts ):-!,member(F,Facts),test_if(Fs,Facts).
test_if(   F,   Facts ):-  member(F,Facts).
check(Fact,Facts):-not member(Fact,Facts).

rule((produce <= green)).
rule((delicacy <= packed)).
rule((turkey <= lbs15, perishable)).
rule((perishable <= refrigerated)).
rule((perishable <= produce)).
rule((staple <= inexpensive, lbs15, not_perishable)).
rule((watermelon <= produce, lbs15)).
```

図１：ルール・インタープリタ方式の前向き推論プロダクション・システム

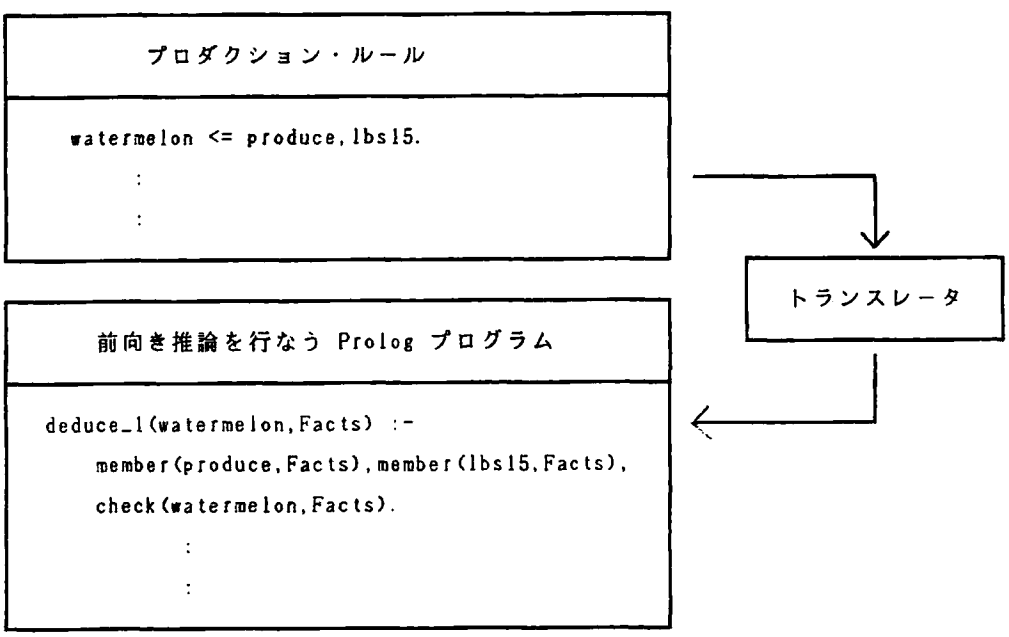

図２：ルールのトランスレート

```
deduce(Facts,Final_Facts):-
        deduce_1(Conseq,Facts),deduce([Conseq|Facts],Final_Facts).
deduce(Final_Facts,Final_Facts).
check(Fact,Facts):- not member(Fact,Facts).

deduce_1(produce,Facts):-
        member(green,Facts),check(produce,Facts).
deduce_1(delicacy,Facts):-
        member(packed,Facts),check(delicacy,Facts).
deduce_1(turkey,Facts):-
        member(lbs15,Facts),member(perishable,Facts),check(turkey,Facts).
deduce_1(perishable,Facts):-
        member(refrigerated,Facts),check(perishable,Facts).
deduce_1(perishable,Facts):-
        member(produce,Facts),check(perishable,Facts).
deduce_1(staple,Facts):-
        member(inexpensive,Facts),member(lbs15,Facts),
        not member(perishable,Facts),check(staple,Facts).
deduce_1(watermelon,Facts):-
        member(produce,Facts),member(lbs15,Facts),
                        check(watermelon,Facts).
```

図３：ルール・トランスレート方式の前向き推論プロダクション・システム

3．前向き推論と後ろ向き推論の併用

3.1 併用の方法

ある命題を証明する際、その命題を結論とする推論規則が複数あるとする。その場合、すでにわかっている事実のひとつに注目し、その事実を前提部に含む規則をその中から選び、前提部の残りの事実を証明してやれば目的の命題が証明される。

3.2 併用システム

Prologによる構文解析システムＢＵＰの制御構造を応用して、上のような推論を行なうプロダクション・システムを考案した。図４において、

```
a <= b,c,d.
```

というプロダクション・ルールは、ＢＵＰと同様に、

```
b(G) :- link(a,G),goal(c),goal(d),a(G).
```

のように変換されている。証明したい命題をＧとしてgoal節を呼ぶと、すでにわかっている事実の一つをfactにより取り出し、それを頭に持つruleを呼ぶ。そのとき、目標のゴールＧは、引数に持つ。ruleは、前提部の残りの条件をgoalにより証明し、結論を起動する。起動された結論と、引数に持っていたゴールが同じになった時、停止条件節により、証明が終了する。このように、基本的には、与えられた事実をもとに、ルールを適用していく前向き推論であるが、前提部の２番目以降の条件に対しては後ろ向きの予測を行なっている。

link節は、目的のゴールに結びつかない事実を早期に棄却する。すなわち、

```
a <= b,c,d.
```

のルールについてみると、事実 bはゴール aに結びつく可能性があるので、

```
link(b,a).
```

を作る。さらに、

```
link(X,Y).    かつ    link(Y,Z).
```

を満たすすべてのＸ，Ｚについて、

```
link(X,Z).
```

を作る。ＢＵＰでは、link節は、高速化のために導入されたが、このプロダクション・システムでは、link節は必要不可欠であり、これがないと、無駄な事実を無限に拾い続ける。

3.3 考察

この方法では、常にルールの前提部の先頭の条件をもとに、適用するルールを選び出す。したがって、その条件が成り立っていれば、そのルールが適用できる可能性が大きいような、重要な条件を先頭に持って来る事が有効である。また、このシステムについても、再計算を避ける機構を加える事が必要である。

```
goal(G) :-fact(H),P=..[H,G],call(P).

% rules
green(G)          :- link(produce,G),produce(G).
packed(G)         :- link(delicacy,G),delicacy(G).
lbs15(G)          :- link(turkey,G),goal(perishable),turkey(G).
refrigerated(G)   :- link(perishable,G),perishable(G).
produce(G)        :- link(perishable,G),perishable(G).
inexpensive(G)    :- link(staple,G),goal(lbs15),not(goal(perishable)).
                           staple(G).
produce(G)        :- link(watermelon,G),goal(lbs15),watermelon(G).

% known facts
fact(green).
fact(lbs15).

% terminate clauses
packed(packed).
delicacy(delicacy).
green(green).
inexpensive(inexpensive).
watermelon(watermelon).
perishable(perishable).
refrigerated(refrigerated).
staple(staple).
turkey(turkey).
produce(produce).
lbs15(lbs15).

% link relations
link(green,produce).
link(packed,delicacy).
link(lbs15,turkey).
link(refrigerated,perishable).
link(produce,perishable).
link(inexpensive,staple).
link(produce,watermelon).
link(green,watermelon).
link(green,perishable).
link(X,X).
```

図４：ＢＵＰを応用した、前向き後ろ向き併用プロダクション・システム

4．まとめ

効率的な前向き推論プロダクション・システムと、前向き推論後ろ向き推論併用プロダクション・システムの、Prologでの実現方法を示し、その利点について論じた。このように、Prologは、プロダクション・システムのインプリメント言語としても適当であるので、今回述べたプロトタイプにさらに検討を加え、今後、ルール・トランスレータを作成する。

最後に、プロダクション・システムを Lisp でインプリメントする場合では、通常、バックトラックの難しさが問題となるが、Prologによるインプリメントでは、これに関しても、Prolog自身のバックトラック機能が利用できるため、問題にならない事を指摘しておく。

謝辞

本研究に関し、御討論いただいた、東京工業大学田中研究室の皆様に、感謝します。

参考文献

(1) Clark,K.L. & McCabe,F,G
PROLOG : a language for implementing expert systems
Machine intelligence, 10 (Hayes & Michie eds.)(Ellis and Horwood,1980)

(2) Mizoguchi,F
PROLOG Based Expert System
New Generation Computing,1, 1983

(3) 古川康一
Prologによるプロダクション・システムの記述
知識工学 pp.138-139 田中幸吉編（朝倉書店 1984)

(4) Matsumoto,Y., Tanaka,H., Hirakawa,H., Miyoshi,H., Yasukawa,H.
BUP : A Bottom-Up Parser Embedded in Prolog
New Generation Computing,1, 1983