Prologによるルール型知識表現法とその利用

Rule Oriented Knowledge Representation in Prolog

田中穂積
TANAKA HOZUMI
東京工業大学
Tokyo Institute of Technology

1. はじめに

ルールの集合が与えられ，それに従った推論を行う場合に，ルールの集合から取り出した個々のルールを解釈するプログラム（ルール・インタプリータ）が必要になると考えられている．たとえば，構文解析を行う場合を考えると，ルールの集合を文法規則に，推論を構文解析を進めることに，ルール・インタプリータをパーザに対応させることができる。

ルールの形式をホーン節に限定し，推論方式を，縦型（depth-first）で後向き（backward）に限定すれば，ルール・インタプリータをPrologインタプリータで代用することができる。したがって特別にルール・インタプリータを作成する必要がない．[1] ところが，推論の方式を前向き（forward）にすると，Prologの推論方式と異なるため，Prologであってもルール・インタプリータが必要であるとされてきた。この場合には，ルールを単位節（unit clause）の引数として埋め込み，それをインタプリートすることになるが，後向き推論に比べて，Prologの良さを生かしきれない．[2] 前向き推論であっても，Prologの良さをフルに生かすことはできないだろうか．プロダクション・システムを例にとり，それが可能なことを2章で示す．

自然言語の意味処理を行なう時，辞書項目の記述はデータとして扱われ，時たまそれに付加した手続きを起動すると考えられてきた．辞書項目をフレームとして見なす場合には，これはリスト構造として表現するのが適切であるとされてきた．このリスト構造には，スロットに対応したリストを（要素として）含んでいる。意味解析は，スロットを次々にたどり，スロットに書き込まれた意味制約条件を検査する．フィラが，全てのスロットの意味制約条件を満たせなければ，上位の辞書項目のスロットを調べる。後者は，知識の遺伝として良く知られている操作である．

以上の意味解析の手順を観察すると，

(i) 非決定的処理が含まれている
(ii) 意味制約条件は，論理式で表現できる
(iii) 知識の遺伝はユニフィケーションにより，ゴールをサブゴールに展開する過程としてシミュレートできる．

(i)～(iii)を，Prologの基本計算機構で代用することはできないだろうか。それが可能であれば，意味解析用に作成すべき（辞書項目の）フレーム・インタプリータが不必要になる。意味解析に用いる辞書項目（フレーム）を（リスト構造でなく）ホーン節として表わすことにより，フレーム・インタプリータを作成することなく，意味解析を行うことが実は可能なことを3章で示す．それにより，Prologの基本計算機構をそのまま利用した意味解析が可能なことを示す．

2. 前向き推論プロダクション・システム

筆者等はすでに，DCGで書かれた文法規則を変換して，Prologプログラムを生成し，それを実行することで，ボトムアップに構文解析，意

味解析を行う方式(BUP)を提案している。[3] ボトムアップが前向き推論であることを考えると，この方式が，前向き推論を行なうプロダクション・システムにそのまま適用可能であることが分かる。比較のために図1に，ルール・インタプリータ方式による前向き推論プロダクション・システムを，図2に ルール変換方式による前向き推論プロダクション・システムの基本的な考え方を示す。図3は，ルール変換後のPrologプログラム（ヘッドが述語ruleになっている）で，最もナイーブなバージョンを示す。

図3のPrologプログラムは，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ので，効率が良い。ところが，推論が一段進む毎に，現在分っている事実集合（Facts）全体から事実を一つ選び出し，それをもとに次に適用可能なルールを探している。ルールの適用は，次に利用す

```
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

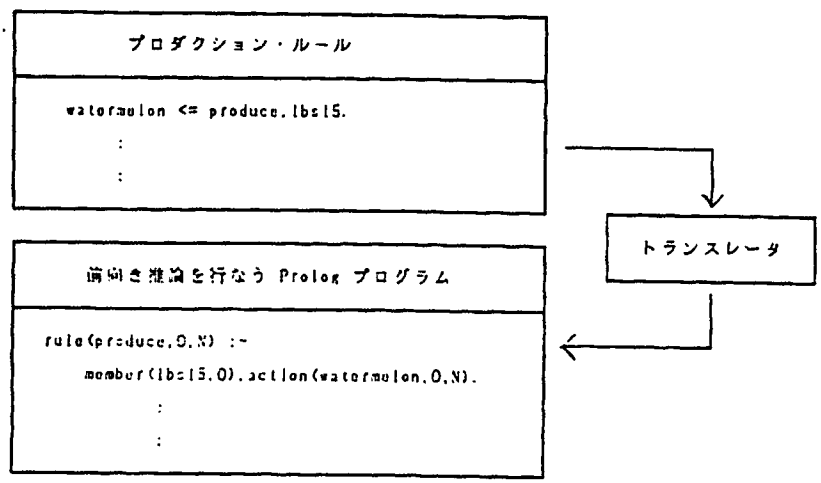

図2：ルールのトランスレート

```
deduce(Facts,Final_Facts):-goal(Facts,Facts,Final_Facts).

goal([],A,A) :- !.
goal([H|T],Old_STM,Final_STM) :-
        ( rule(H,Old_STM,Final_STM) ; goal(T,Old_STM,Final_STM) ).

action(A,B,Final_STM) :- not(member(A,B)),goal([A|B],[A|B],Final_STM).

rule(green,O,N)         :- action(produce,O,N).
rule(packed,O,N)        :- action(delicacy,O,N).
rule(lbs15,O,N)         :- member(perishable,O), action(turkey,O,N).
rule(refrigerated,O,N):- action(perishable,O,N).
rule(produce,O,N)       :- action(perishable,O,N).
rule(inexpensive,O,N) :- member(lbs15,O),not(member(perishable,O)),
                         action(staple,O,N).
rule(produce,O,N)       :- member(lbs15,O), action(watermelon,O,N).
```

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
（第0バージョン）

```
goal(G) :-fact(H),link(H,G),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(_,G):-var(G),!.
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を応用した効率的な前向き推論プロダクション・システム
（第１バージョン）

べき事実集合の範囲を限定する．これはルール適用結果として得られるトップダウン的な予測情報と見なせるが，図３のPrologプログラムでは，それを利用する手だてがない。文献(3)のBUPでは，この予測を用いて，トップダウンとボトムアップ法とを融合した効率の良い構文解析を行うPrologプログラムに変換している。実際，文献(3)で述べた方式を，そのまま前向きプロダクション・システムに応用できる(4)。それを図４に示す。

図４では，

a <= b, c, d

というプロダクション・ルールを

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

のように変換する。

Gは，推論の結果帰結される命題であり，前向きに推論している過程では変数として与える。ルールの適用により，トップダウンの予測が可能になると，Gにはそれが与えられることになる。図４のgoal節は，事実集合から事実を一つ引き出し（fact(H)），それがGを満たす可能性があるかどうかをlink(H,G)で調べた後に，call(P)により，引き出された事実をヘッドに持つ節をよぶ。

link節は，ルールが与えられた段階で次のようにして計算できる。

a <= b, c, d

というプロダクション・ルールでは，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). & link(Y,Z) を

満たすすべてのX，Zについて

link(X,Z).

を作る．また全てのXについて，

link(X,X).

Gが変数でなければ，linkにより，goal節のボディでGに結びつく可能性のある事実のみを拾い，またルールの帰結がGに結びつく可能性がある時にのみ，そのルールを試みる。したがって，プロダクション・システムがGに結びつかない推論を無限に繰り返すことを避けることができる。

以上の議論から，図4に示したものは，最初Gを変数として前向き推論を行っても，ルールの適用が一たび行なわれると，以後は，サブゴールを予測したトップダウンとボトムアップが併用された推論が行なわれるので，効率が良いことが理解できよう。さらに，図4はコンパイルすることにより，一層の高速化をはかることができる。

3. ホーン節による辞書項目の表現

図5に，ホーン節によるopenの辞書項目の表現を示す。それによると，フレームを構成するスロットが述語として，またそれらが，;で結合されていることが分かる。それにより，スロットの非決定的な選択が，Prologの基本計算機構で代用できることが分かる。

スロットの意味的制約条件は，述語木の任意の論理結合であらわされており，意味制約条件の検査は，単にそれらをcallするだけでよい。self述語が最後にかかれているが，これは，フィラーと，self以前のスロットに書かれた意味制約条件とが整合しないばあい，self述語の第2引数act(…)がself節のボディでよばれて，actをヘッドとする節とのユニファイが起り，actのボディにあるスロットが調べられる。これは，ユニフィケーションによるinheritance of knowledgeが実現されたことを意味している。

以上の基本的な考え方により，意味解析用プログラムがほとんど皆無に近くなることを筆者は確認している。今後，システムを大規模化することを検討している。

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) 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)

(3) Matsumoto, Y,
BUP : A Bottom-Up Parser Embedded in Prolog
New Generation Computing, 1, 1983

(4) 山本聡、田中穂積
Prologによるプロダクション・システムに関する一考察
日本ソフトウェア科学会・第1回大会、1984


```
1   v((open:UNIT)) --> (open).
2   open(UNIT,OLD_PREP,N,SNAM) :-
3                   UNIT= (SUBJ,OBJ:NEW_PREP),
4                   (subj(SNAM,(t(agent,human,SUBJ,N);
5                                 t(obj,thing_open,SUBJ,N);
6                                 t(obj,activity,SUBJ,N);
7                                 t(obj,place,SUBJ,N);
8                                 t(instrument,instrument,SUBJ,N);
9                                 t(reason,wind,SUBJ,N))),NEW_PREP=OLD_PREP;
10                    obj(SNAM,(t(obj,thing_open,OBJ,N);
11                              t(obj,activity,OBJ,N);
12                              t(obj,place,OBJ,N))),NEW_PREP=OLD_PREP;
13                    with(SNAM,t(instrument,instrument,PREP2,N)),
14                        prep_check(OLD_PREP,NEW_PREP,PREP2);
15                    self(SNAM,act(UNIT,OLD_PREP,N,SNAM))).
```

図5 ホーン節による辞書項目の表現

(constraintのチェックは、t(deep_case,constraint,SNAM,FILLER) がCALLされることにより行なわれる。(1) t(deep_case,constraint,SNAM,FILLER)は、constraintとFILLERが一致するか、(2) あるいはFILLERの上位がconstraintと一致したときに、(3) (すなわち、FILLERの上位からselfがCALLされ(3) にマッチする) 、SNAM=(deep_case,FILLER) という処理を行なって値を返す。
inheritance of knowledgeは、self(SNAM,act(UNIT,OLD_PREP,FILLER,SNAM)) をCALLすることで行なう。selfというpredicateは、constraintのチェックでも使われ、両者で共用している。)