Prologによる概念依存型構文解析

浅見　徹
（KDD研究所）

1. はじめに

Shankの概念依存文法による自然言語処理へのアプローチ[Birnbaum 81]は、言語理解レベルを単文理解から文脈理解へ拡張する際に貴重な手法を提供する。ここでは、従来主としてトップダウン処理に用いられてきたProlog上に概念依存文法をインプリメントすることを試みている。筆者がUNIX 4.1bsd上で使用しているCProlog[Pereira 82]にはトップダウン・パーザDCG（definite clause grammar）[Pereira 80]が組み込まれており、高速かつ柔軟な構文解析が可能であるため、自動分かち書き処理機能等を付加して日本語処理に用いてきた[浅見 83]。しかし、概念依存型の構文解析では文法規則を単語辞書中にリクエストという形態で登録し、一種のプロダクション・システムによりパーズをボトムアップに行うというアルゴリズムをとっているため、DCGで記述するには無理がある。本稿で提案するパーザはリクエストをPrologのクローズに対応させ、リクエストのactivate, deactivateをデータベースの書き換えに対応させることにより、ボトムアップ処理を実現している。パーザをProlog上に実現した結果、Prologのunificationをslot-fillingに利用できること、上位レベルで自然な形で推論機構を導入することができること等の利点がある。Prologを用いたボトムアップ・パーザとしては、Bup[松本他 82]などがあるが、シンタックス主導型であるため不完全文の処理に関して問題がある。本パーザは処理速度の面では劣るが、不完全文の処理は自然にできる。

2. Prologによる概念依存文法の記述方法

最初にCA[Birnbaum 81]に代表される概念依存型の構文解析のアルゴリズムを簡単に紹介する。この型の構文解析では、処理途中に生成される概念構造のために短時記憶C-LIST，アクティブ・リクエスト収容のためにR-LISTをデータ構造として持っている。処理はアクティブ・リクエストを評価することによりボトムアップに進行する。各リクエストはtest機能とaction機能を持つ一種のプロダクション規則であって、それぞれ次に示す機能を持つ。

test機能：
(1) 特定の単語や句の出現をチェックする。
(2) C-LISTに特定の概念構造があるか否かをチェックする。
(3) C-LIST中の概念構造の特定のギャップが満たされているかをチェックする。

action機能：
(1) C-LISTに概念構造を追加する。
(2) 概念構造の中のギャップを他の概念構造で満たす。
(3) 他のリクエストをR-LISTに登録する（activateする）。
(4) リクエストをR-LISTから削除する（deactivateする）。

2.1. 概念構造のProlog表記について

以上の前提のもとで、パーザをProlog上に実現する場合、先ず概念構造の表記法を確立しなければならない。特にリクエストのaction機能の(2)の場合、再起的な埋め込みになることがないようにしなければならない。本稿では、Shankの用いているリストではなく[Birnbaum 81]、述語を用いている。例えば、概念構造を下図のように表記する。

```
(PP CLASS (VEHICLE) TYPE (AIRPLANE) SIZE (SMALL))  ..... Shank流[ 81]
 pp(i), class(vehicle), type(airplane), size(small)  ..... Prolog表記
```

図1. "小さい飛行機"に対応する概念構造

ここで、iは文の処理中に現われる概念構造に、出現順に付けられた番号を示す。この表記法により、action機能(2)で無限大の構造を生成することを避けることが可能である。図2に[Birnbaum 81]の中からとった"1500ポンドのマリファナを積んだ小さな飛行機"に対応する構造を示す。

```
pp(1), class(vehicle), type(airplane), size(small), ref(indef),
      rel( ptrans(2), actor(A),
                  object( pp(3), class(physobj), type(marijuana),
                                 amount( unit(type(lb),
                                              number(num(val(1500))))),
                  to(inside(part(pp(1)))))
```

図2. "1500ポンドのマリファナを積んだ小さな飛行機"の概念構造

図2のppには全て一意的に番号が割り当てられているため、Prologデータベースに登録された概念構造の中にある図2に示す構造は、pp(1)によって一意に指すことができるため、inside(part(X))のギャップ・フィリングにpp(1)を用いることができる。この埋め込み方法により、action機能(2)で無限大の構造を生成するのを避けられることが分かる。

2.2. リクエストのProlog表記について

リクエスの表記は下記に示す述語reqによる。

req(i, Ip, Clist, Nclist, Cwords, Nwords, Test, Action) ...(1)

ここで、iは辞書登録時に各リクエストに一意に割りあてられる番号，Ipは(1)のリクエストがactivateされた時に割りあてられる番号であり、このリクエストがActionにより概念構造をClist中にロードする際の位置を示す。ClistおよびCwordsはそれぞれリクエスト評価前のC-LISTおよび入力文字列を示し、NclistおよびNwordsはリクエスト評価後のC-LISTと入力文字列を示す。TestおよびActionは2節で示したTestおよびAction機能を実現するためのProlog述語の列である。図3に"小さい"に対応するリクエストを示した。ここで、リクエストの辞書中の番号を5，該Actionの結果概念構造をClistの2番目の位置にロードするものとする。Cwords = Nwords = Words はこのリクエストが入力文字を読まないことを示している。

```
reg(5, 2, Clist, Nclist, Words, Words,
        find(pp,(pp(K), Rest), Clist, Pn, Pn>2), /* Clist上で"小さい"の後のppを見つける */
        chpool(Clist, Pn, (pp(K), size(small), Rest), Nclist),
    /* 見つけた位置Pn上の値(pp(K), Rest)にsize(small)を付加し、次のC-LISTであるNclistを作成 */
        deactivate(5).  /* 自分自身をPrologデータベースから削除する */
```

図3. "小さい"に対応するリクエスト

2.3 リクエストのactivateとdeactivate

リクエストのactivateとdeactivateはPrologデータベースへの登録および削除を行えばよく、図4で定義できる。

```
activate(Ip, reg(I, NIp, Clist, Nclist, Cwords, Nwords, Test, Action)):-
        NIp is Ip+1,
        asserta(reg(I, NIp, Clist, Nclist, Cwords, Nwords, Test, Action)).
deactivate(Ir):- retract(reg(Ir,_,_,_,_,_,_,_)).
```

図4. activateとdeactivateのProlog表記

2.4 辞書記述について

辞書項目の各々は述語dictionaryに収められており、以下に示す形式をとる。

```
dictionary(word, wordclass, Ip, Requests, [word|Rest], Rest).        -----(2)
```

ここで、[word|Rest]は現在の文字列がWordから始まることを示し、wordclassはwordの品詞名、Ipは(1)の場合と同様であり辞書引き時のIpがバインドされる。また、requestsは2.3のactivate(Ip, reg(I, NIp, Clist, Nclist, Cwords, Nwords, Test, Action))の形式を持つProlog述語1個以上から構成されている。

2.5 制御プログラムの記述について

制御プログラムの動作は、[Birnbaum 81]によると次の通りである。

(1) 次の単語を入力文字列から抽出する。(辞書引き。)なければ処理終る。
(2) リクエスト評価環境(normal mode か nounphrase mode)を決定する。
(3) 特定の単語を予測しているリクエストがあれば、それを評価する。goto (5).
(4) リクエストを辞書からロードする((2)のrequests部を評価する。)
(5) アクティブなリクエストを(2)で定めた環境で評価する。
(6) ステップ(1)へgoto。

上記の動作を行うPrologプログラムを図5に示す。

```
parse(Input, CD, Ip, Prev_wordclasses, Prev_env, Clist):-
    dictionary(Word, Wordclass, Ip, Requests, Input, Rest),
    decide_env(Prev_env, New_env, Prev_wordclasses, Wordclass, Wordclasses),
    (consider_requests_for_words(Ip, Clist, Nclist, Word, NIp);
     load_dictionary(Ip, NIp, Requests, Clist, Nclist)),
    consider_requests(New_env, NIp, Nclist, Nlist),
    parse(Rest, CD, NIp, Wordclasses, Newenv, Nlist).
parse([], CD, _, _, _, Clist):-
    getresult(Clist, CD).
```

図5. 制御プログラム parse

ここで、parseの引数Inputは入力文字列であり、初期値は全入力文字列である。CDはparse終了後に抽出される概念構造、Ipは現在のClistの位置、Clistは現在まで得られた概念構造のリスト，Prev_envは直前の解釈環境（normalまたは、noun_phrase）、Prev_wordclassesはnoun_phraseモードの場合直前までの単語カテゴリのリストであり、normalの場合は[]である。初期値は、Ip＝φ，Clist=[]，Prev_env＝normal，Prev_wordclasses ＝[]である。単語カテゴリについては、[Gershman 77]を参照している。decide_envは過去の解釈環境Prev_envと単語カテゴリリストPrev_wordclasses，入力単語カテゴリWordclassから次の解釈環境New_envと単語カテゴリWordclassesを求める。consider_requests_for_wordsはWordを予測しているリクエストを調べ、あれば評価し、新らしい概念構造リストNclistを作成する。lord_dictionaryは辞書中のリクエストをPrologデータベースに登録する（ただし、Clist=Nclist）。consider_requestsは環境New_envでPrologデータベース中のリクエストを評価し、新らしい概念構造リストNlistを求め、次のparseへ渡す。getresult(Clist, CD)は概念構造リストClistの中の最終概念構造にCDをバインドするものである。

[例]

図6(a)の辞書で、"john goes."を処理した例を図6(b)に示す。パーザへのコールは、次式(3)で始まったものとする。

```
| ?- parse([john,goes,.], CD, φ, [], normal, []).        ……(3)
```

```
dictionary(john,noun,Ip,
          activate(Ip,req(1,NIp,Clist,Nclist,Cwords,Cwords,true,
                          (chpool(Clist,NIp,(pp(1),person(name(john))),
                                  Nclist),
                           deactivate(1))),
          [john|Rest],Rest).
dictionary(goes,verb,Ip,
          activate(Ip,
                   req(2,NIp,Clist,Nclist,Cwords,Cwords,true,
                       (chpool(Clist,NIp,
                               (ptrans,actor(A),object(A),to(B),
                                from(C)),Nclist),
                        activate(NIp,
                                 req(3,NNIp,Nclist,NNclist,Cwords,Cwords,
                                     find(pp,(pp(K),Rest),Nclist,Pn,(Pn<NIp)),
                                     (chpool(Nclist,Pn,Var,Temp),
                                      chelement(Nclist,NIp,actor(A1),
                                                actor((pp(K),Rest)),NNclist),
                                      deactivate(3)))),
                        deactivate(2)))),
          [goes|Rest],Rest).
```

図6(a) "john goes."のための辞書例

ここで、図6(b)の左端数字は、図5に示した各述語の実行順序を示している。各述語の引数に関しては、(9)のNlist，(10)のCD以外はバインドされた値を代入してある。また、(1)および(3)の"..."は図6(a)の"john"に対するリクエスト部を、()および(8)の"..."は図6(a)"goes"に対するリクエスト部を示している。

(9)のchelementの意味は、Nclistの2番目の要素でactor(A1)とマッチングするものを見つけ、A1に対応する変数に(pp(K), Rest)をバインドし、新らしいC-LISTNNclistを作成することを意味する。Prologのパターン・マッチングの性質によ

```
| ?- parse([john,goes,.],CD,0,[],normal,[]).
(1) dictionary(john,noun,0,...,[john,goes,.],[goes,.])
(2) decide_env(normal,noun_phrase,[],noun,[noun])
(3) load_dictionary(0,1,...,[],[])
   Prolog Database: req(1,1,Clist,Nclist,Cwords,Cwords,true,
                        (chpool(Clist,1,(pp(1),person(name(john))),Nclist),deactivate(1))).
(4) consider_requests(noun_phrase,1,[],[[(pp(1),person(name(john)))]])
   Prolog Database: No requests
(5) parse([goes,.],CD,1,[noun],noun_phrase,[[(pp(1),person(name(john)))]])
(6) dictionary(goes,verb,1,...,[goes,.],[.])
(7) decide_env(noun_phrase,normal,[noun],verb,[])
(8) load_dictionary(1,2,...,[[(pp(1),person(name(john)))]],[[(pp(1),person(name(john)))]])
   Prolog Database: req(2,2,Clist,Nclist,Cwords,Cwords,true,
                        (chpool(Clist,2,(ptrans,actor(A),object(A),to(B),from(C)),Nclist),
                         activate(2,
                                  req(3,NNIp,Nclist,NNclist,Cwords,Cwords,
                                      find(pp,(pp(K),Rest),Nclist,Pn,(Pn<2)),
                                      (chpool(Nclist,Pn,Var,Temp),
                                       chelement(Nclist,2,actor(Al),actor((pp(K),Rest)),NNclist),
                                       deactivate(3)))),
                         deactivate(2))).
(9) consider_requests(normal,2,[[(pp(1),person(name(john)))]],Nlist)
    execute request 2 then
        Prolog Database:
                    req(3,3,Nclist,NNclist,Cwords,Cwords,
                        find(pp,(pp(K),Rest),Nclist,Pn,(Pn<2)),
                        (chpool(Nclist,Pn,Var,Temp),
                         chelement(Nclist,2,actor(Al),actor((pp(K),Rest)),NNclist),
                         deactivate(3))).
        C-LIST = [[(ptrans,actor(A),object(A),to(B),from(C))],[(pp(1),person(name(john)))]]
    execute request 3 then
        Prolog Database: No requests
        Nlist = [[(ptrans,actor((pp(1),person(name(john)))),
                          object((pp(1),person(name(john)))),to(B),from(C))],Var]
(10) parse([.],CD,3,[],normal,
              [[(ptrans,actor((pp(1),person(name(john)))),object((pp(1),person(name(john)))),
                        to(B),from(C))],Var])
(11) getresult([[(ptrans,actor((pp(1),person(name(john)))),object((pp(1),person(name(john)))),
                         to(B),from(C))],Var],
          (ptrans,actor((pp(1),person(name(john)))),
                  object((pp(1),person(name(john)))),to(B),from(C)))
CD = (ptrans,actor((pp(1),person(name(john)))),object((pp(1),person(name(john)))),to(B),from(C))
```

図6(b). "john goes." の処理例

り、NNclistのptransのobjectスロットにはactorスロットを埋めている値と同じものがバインドされるから、スロット・フィリングに関してはLispを用いて実現した場合よりも柔軟に行えられる。

3. 検討課題

この実現方法は、C-LISTを実現するのにリスト構造を使用していること、各リクエストの定義全てを引数にバインドする場合があること、インタプリタであること等の理由により、処理速度は速くない。一方、DCGの場合、文法規則は予めPrologのクローズにコンパイルされているため、処理速度は速いが、不完全文の処理を目差すと文法規則が増加し、処理が無限ループに陥らないことを検証することが困難になる。筆者の目標は会話文の理解を目差しているが、このような場合には、不完全文の処理は不可避の問題である。本稿の目的は、不完全文の自然な解析が可能なボトムアップ型パーザ(概念依存型構文解析)をProlog上に実現することにより、Prologのトップダウンの推論機構と融合させ、より柔軟なシステムを実現することにある。

Prologを用いて実現した利点は次のとうりである。

(1) スロット・フィリングをPrologのunificationを用いて自然な形で実行できる。

(2) リクエスト評価規則(recency rule)は、activateを図4で定義してあるため、Prologのデータベース評価順序に一致し、リクエストはactivateされた

時と逆順に評価されるため、新うたに評価用のプログラムを書く必要はない。また、解決すべき課題としては、次の2点が挙げられる。

(a) 単語の意味があいまいなために、(i)式で I_P が同じリクエストが2個以上真になる、いわゆるdisambiguationの問題を解くために、[Birnbaum 81]では次のようなヒューリスティックスを導入している。

(i) Testが真となるリクエスト（同一I_P）を集め、1つしかないときはそれを評価、真になるリクエストがなければStop. そうでないときはgoto (ii)。

(ii) リクエストがC-LISTに付加する概念構造のスロットが他の概念構造で埋められるか、または付加する概念構造が他の構造のスロットを埋められるようなリクエストを集める。

(iii) 1つも(ii)に合致するリクエストがなければStop, 1つあるときはそれを実行、複数あるときは評価を延期し、best-connectedを埋め込みをするリクエストが判明したらそれを評価し、残りのリクエストをdeactivateする。

Prologでこのような手続き的処理を実現しようとすると、[Kahn 82]がactorを実現した例から推察されるように、非常に能率が悪い。一つの解決策としては、図4のようなデータベースの書き換えをやうずに処理系を実現し、Prologのbacktrackingを利用して全ての可能性を尽くす方法がある。もう一つの解決策はProlog自体にLispのprog関数に相当する処理ができるようにする方法である。幸い筆者の用いているCPrologはCで記述された比較的コンパクトなインタプリタであるため、後者のアプローチをとっている。

(b) 日本語処理を行えるためには、Gershmanのnoun group boundaryヒューリスティクスに相当する評価関数を定める必要がある。特に日本語の場合、用言の語尾変化、助動詞の接続規則のようにシンタクスのみで処理可能な部分は、辞書引き時に処理してしまう方が望ましい。従って、dictionaryはDCGあるいはBUPを用いて書き直す必要がある。評価関数はその後に決めた方がよい。この変更を行った後の概念依存型の構文解析は係り受けによる日本語の文法規則にかなり近いものになると考えられる。

4. まとめ

概念依存型構文解析をProlog上に実現するとの観点から再構築することを試みた。Prolog上に実現したことにより、スロット・フィリングが自然に行えること、リクエスト評価順序とPrologのデータベース評価順序が一致するなどの利点があるが、処理速度の向上、特に高速なdisambiguationプロセスをインプリメントすることが望まれる。また日本語処理のためには、語尾処理用の専用プログラムを作成することが望まれるが、リクエスト評価を制御する規則の確立と合わせて現在検討中である。

5. 謝辞

日頃御指導を戴く鎌治所長、寺村副所長、深田次長、榑松室長並びに端末装置研究室の各位に感謝する。また、CPrologを提供して下さったエジンバラ大学の関係各位に感謝致します。

参考文献

[Birnbaum 81] Birnbaum, L. and Selfridge, M., "Conceptual Analysis of Natural Language", Inside Computer Understanding (Ed. Schank, R. C. and Riesbeck, C. K.), Lawrence Erlbaum Associates, pp. 318-353, 1981.

[Pereira 82] Pereira, F., "CProlog User's Manual Version 1.1", Dept. of Architecture, Univ. of Edinburgh, 1982.

[Pereira 80] Pereira, F. and Warren, D., "Definite Clause Grammar for Language Analysis - A Survey of the Formalism and a Comparison with Augumented Transition Networks", Artificial Intelligence, 13, pp. 231-278, May, 1980.

[浅見 83] 浅見徹, "シンタックスを考慮したバッファを持つ概念依存型構文解析", 情報処理学会第26回全国大会, 5L-7, 1983.

[松本 82] 松本裕治, 田中穂積, "Prologに埋め込まれたbottom-up parser: BUP", 情報処理学会自然言語処理研究会, 34-6, Dec., 1982.

[Kahn 82] Kahn, K.M., "Intermission - Actors in Prolog", Logic Programming (Ed. Clark, K. L. and Tärnlund, S.-A.), Academic Press, pp. 213-228, 1982.

[Gershman 77] Gershman, A. V., "Conceptual Analysis of Noun Groups in English", Proc. of the 5th IJCAI, Cambridge, Mass., pp. 132-138, 1977.