辞書のＴＲＩＥ構造化と熟語処理

Idiom Handling By TRIE Sturucture Dictionary

上脇　正　（東京工業大学）
田中　穂積（東京工業大学）

Abstract　Matsumoto et al. developed a Bottom-Up parser embedded in Prolog (BUP) which can handle CFG with left recursive rules. In the BUP, some types of idioms have been treated as grammar rules. But it seems to be obvious that as the number of grammar rules are increased, the parsing speed are decreased. In this paper, we describe the other idiom handling method in which idiom are directly put in a dictionary. According to our idiom processing method, which is embedded in the new BUP, we can analyse an idiom like 'not only ~ but also ~' easily, since non-terminal symbols and Prolog predicates are allowed to be in a dictionary entry. As our dictionary consists of a data sturucture called TRIE, common words in the dictionary (for example, 'computer' in 'computer network' and 'computer system') can share the same memory space and we can not only save memory space for a dictionary but also improve parsing speed.

１．序論

今日広く使われているDEC-10 Prolog やC-Prologには文法記述を柔軟に行なう形式としてＤＣＧ(Definite Clause Grammar) [Pereira 80] が埋め込まれている．ＤＣＧで記述された文法と辞書は、Prologプログラムに変換され、それを直接実行することで（パーザを作成することなく）トップダウン法による構文解析を高速に行なうことができる．しかし、Pereira 等のこの方法では解析をトップダウンで縦型探索により進めていくため、左再帰的な文法規則が扱えないという欠点がある．そこで、Pereira 等の方法の利点を損なうことなく、ボトムアップにパーザを行なうシステムが松本［松本 82, 83］等により開発された．これはＢＵＰ(Bottom Up Parser in Prolog)と呼ばれている．ＢＵＰはＤＣＧで記述された文法を一度ＢＵＰトランスレータによりボトムアップ的な規則のＢＵＰ節に変換し、それをPrologプログラムに変換して実行する．

今回、ＢＵＰにおける熟語処理の新しい方式を考案したので、その有効性をメモリ効率、処理速度、記述の容易性の観点から検討した結果について述べる．熟語は複数個の単語の系列であるため、熟語を処理する際には次のような問題がある．

・どこまでが熟語か判断することが難しい．たとえば、pick up という簡単な熟語でもpick it upという形がある．
・not only ~ but also ~ のように、~の部分に不特定で文法的に同一の機能を持つ語や句が来る熟語をどのように扱うか．
・熟語の中の単語の語形変化をどう処理するか．(get up →getting up)
・computer system などの名詞の並んだものも熟語と考えれば、熟語の数は膨大なものになり、それらを一つ一つのHorn節で登録していたのでは、メモリの節約にならない．

ＢＵＰに於ける熟語の処理については、［松本　84］の方法があるが、この方法では

・メモリの効率
・辞書引きの速度
・語形変化処理
・通常の辞書の他に熟語用の特別な辞書が必要
・熟語の表記の容易性

の点で十分とはいえない．

本稿の方法は、辞書を、共通の単語が共有できるように木構造（TRIE構造）化し、か

つ辞書の中に非終端記号や補強用プログラムを埋め込めるようにするものである．木構造化により膨大な数の熟語を効率よく辞書に組み込むことができ、辞書引きも無駄なく行なうことができる．辞書の中に終端記号の他に文法カテゴリや補強用のプログラムを付加可能なので、not only ~ but also ~ などの熟語も記述が可能になる．また、熟語はＤＣＧで記述すればトランスレータにより木構造辞書に変換されるので、熟語の表記が非常に容易である．

ＢＵＰの熟語処理の実行環境は、SUN2/120 Work StationのUNIX4.2 bsd 上で動くC-Prologインタープリタである．

2．辞書のTRIE構造化の利点

従来のＢＵＰの辞書では、Fig.2.1 に示す語を登録するために、Fig.2.2 のように１語につき１つのdict節が必要であった．

```
n --> [computer] .
n --> [computer, interface] .
```

Fig.2.1 Examples of DCG's dictionary

```
dict(n, [] ) --> [computer] .
dict(n, [] ) --> [computer, interface] .
```

Fig.2.2 Examples of original BUP's dictionary

この場合computerが２つの節にでてきて、メモリが無駄であり、辞書引きにも時間がかかる．そこで、重複する単語は共有できるような構造を持つ辞書を考えた．この構造をTRIE構造［Aho 81］と呼ぶ（TRIEはreTRIEval からきており、元来は単語を一字毎に分けて木構造にするものであった［奥村84］．）

次に、これまで文法規則として記述されていた熟語を木構造辞書に組み込み、辞書引きの過程でこの種の熟語処理を行なうアルゴリズムを開発した．英和辞典を見ても分かる通り、この種の熟語は辞書項目の中に入れるのが自然に思える．そこで、辞書をTRIE構造化するにあたり、次の事を可能にした．

・熟語の中に文法カテゴリを埋め込む．
　(not only (np) but also (np) のような熟語が辞書項目に書ける．)
・熟語の中に補強用のPrologプログラムを埋め込み、詳細な熟語の処理を行なう．

3． TRIE 構造

TRIE構造の辞書とは木の形をした辞書のことである．Fig.3.1 のＤＣＧ節をTRIE構造の辞書に入れると、Fig.3.1 の下に示す構造になる．辞書を木構造にする場合、辞書全体を一つの木にまとめると、辞書に入れる語数が多くなるとその大きな木を受け渡すの

に時間がかかり効率が悪い。そこで、先頭の単語が同じ語同士で一つの木を形成するようにしてある。

根の、t1－［c1］は、終端記号のt1がc1のカテゴリに属する事を表わす。その次のt2－［c2］は、終端記号列t1,t2 がカテゴリc2に属することを表わす。以下、同様である。中心の枝に［c4］－［］と(prog)－［c3］があるが、文法カテゴリや補強用プログラムも［］や（）で囲むことにより単語と同じように扱うことができる。

また、各語にリンク情報を持たせることにより、got の木からget の木を参照するようなことが可能である。

```
c1 --> [t1] .
c2 --> [t1,t2 ] .
c3 --> [t1] ,c4, [t3] , (prog).
c5 --> [t1,t4,t5] .
c6 --> [t1,t2,t6] .
```

図9 → 8

図10：図9の → 9

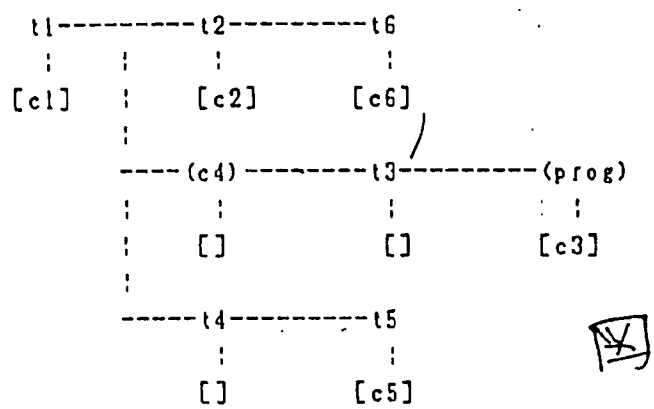

図11 → 10

Fig.3.1 TRIE structure

4．従来の熟語処理方法

従来のＢＵＰでは、非終端記号やPrologプログラムを含む熟語はfigure 4.1のように文法規則として扱っていた。先頭の終端記号［not,only］だけとって、これを新しい文法カテゴリterminal1 として辞書にいれておき、残りは［not,only］をterminal1 にかえて、文法規則として変換している。これでは、実際に存在しない文法カテゴリを作ってしまううえに、熟語を文法規則として扱っておりよくない。

```
dict(terminal1,[]) --> [not,only].

terminal1(Goal,[],Info) --> {link(np,Goal)},
        goal(np,[[NP1_A],_]),
        [but,also],goal(np,[[NP2_A],_]),
        {check(NP1_A,NP2_A,A)},
        np(Goal,[[],[A],[]],Info).
```

Fig.4.1 Original BUP's grammar for 'not only ~ but also ~'

そこで、[松本 84] は、単語用の辞書とは別に熟語用の特別の辞書に熟語を登録する方法を考案した。これは、figure 4.2のように熟語をidiom という節で登録しておき、辞書引きのときに単語の辞書とは別にこれを見にいくわけである。

```
idiom(not,Goal,_,[Syn,Sem,[Goal,[not,not],[only,only],
                  Tree1,[but,but],[also,also],Tree2]]) -->
    [only],{member(Goal,[np,adjp,pp]) },
    goal(Goal,[Syn1,Sem1,Tree1]),[but],[also],
    goal(Goal,[Syn2,Sem2,Tree2]).
```

Fig.4.2 Idiom dictionary of Matumoto's system

今回の、TRIE構造化辞書による熟語処理では、松本等の方法に比べて次の点で優れている。

・熟語を単語の辞書と完全に一体化してある。
・松本等の方法では、語形変化のある語は、特別な形で書いておく必要があるが、TRIE辞書では語形変化を自動的に行なう。
・トランスレータの採用により熟語の記述が容易。

5．TRIE構造辞書項目と熟語記述

我々の辞書構造の例をFig.5.1 に示す。

```
dicta(computer,[[n,[[],[c],[]]]],
        [[interface,[[n,[[],[c],[]]]],
                    [manual,[[n,[[],[c],[]]]]]],
         [system,[[n,[[],[c],[]]]]],
         [network,[],
                  [cable,[[n,[[],[c],[]]]]]]]).
```

Fig.5.1 TRIE structure dictionary

述語の第１引数のcomputerは、この述語がcomputerが先頭にくる語を扱っていることを表わしている。第２引数の［［n,［［］、［c ］、［］］］］は、単語computerのもつ付加情報である。第３引数にはcomputerが先頭にくる熟語をTRIE構造で表わしたものが入っている。ただし、根のcomputerは、この節を選ぶのに使うため特にdicta 述語の第１引数としたので、熟語の木の中には入っていない。Fig.5.1 の熟語の部分を、分かりやすく図にするとFig.5.2 のようになる。この部分の一般的な構造はFig.5.3 のようであり、それが再帰的に表われると考えてよい。

Fig.3.1 では、各語の付加情報が文法カテゴリだけであったが、実際には［［n［［］、［c ］、［］］］］などの情報が付いている。各語に付いている情報の最初のnはその語の文法カテゴリである。

```
------------interface                                          1
:       :
:       ----  [ [n, [ [] ,  [c ] ,  [] ] ] ]                  2
:       :
:       -------------manual                                    3
:              :
:              ------ [ [n, [ [] ,  [c ] ,  [] ] ] ]          4
:
------------system                                             5
:       :
:       ----  [ [n, [ [] ,  [c ] ,  [] ] ] ]                  6
:
------------network                                            7
        :
        ----  []                                               8
        :
        -------------cable                                     9
               :
               ------ [ [n, [ [] ,  [c ] ,  [] ] ] ]          10
```

Fig. 5.2 Sturucture of idiom part

```
①  ------単語1
      |   /
      ---付加情報
      |
      ---更に単語が続く場合1    (①と同じ構造)
      |
      ---更に単語が続く場合2    (①と同じ構造)
      .
      .
      .
```

Fig. 5.3 Sturucture of idiom part

```
n( [] ,  [c ] ,  [] ) --> [computer] .
n( [] ,  [c ] ,  [] ) --> [computer, interface] .
n( [] ,  [c ] ,  [] ) --> [computer, interface, manual ] .
n( [] ,  [c ] ,  [] ) --> [computer, system ] .
n( [] ,  [c ] ,  [] ) --> [computer, network, cable] .
```

Fig. 5.4 Dictionary written in DCG

次は、３つ要素から構成されているが、これらはそれぞれ、
１番目は他の辞書項目を参照するための情報
２番目は構文解析のためのアドバイス情報（c は数えられる名詞の意味）

３番目は意味情報
が入る．１番目の情報は後述するように不規則変化動詞などで過去形から原形を参照する場合などに使う．

今まで見て来た通りTRIE構造の辞書を我々が直接書くのは困難である．そのため、ＤＣＧで記述した辞書項目をTRIE構造の辞書に変換するトランスレータを作成した．例えば、computerに関するFig.5.1 の辞書は、Fig.5.4 のＤＣＧ節を変換したものである．右辺が登録する熟語、左辺の述語名がその文法カテゴリ、その引数がその付加情報である．

６．語い項目と熟語のユーザ記述形式

具体例により語項目と熟語ユーザ記述形式を説明する．get に関係する語の登録を考える．トランスレートする前のユーザ記述形式はＤＣＧ（Fig.6.1 ）である．got が(4) と(5) の二箇所あるのは、got にはget の過去形と過去分詞の二つがあるためである．(4) と(5) のgot の節の第１引数にget が入っているのは、got からget を参照するためである．これによりgot up,got on を辞書に入れておかなくても、get up,get on の処理が可能になる．ただし、動詞の変化形は、不規則変化でないかぎり、ＢＵＰが自動的に語形変化処理を行なうので辞書に入れる必要はない（例えばgetting up,getting on など）．形容詞、副詞の語形変化も同様である．(1) のget upの節の最後に付いている”！”は、Prologのカットオペレータと同じ働きをするもので、構文解析のとき、get upまでマッチした後、失敗してバックトラックしてきた場合、他のget に関する辞書引きをさせたくないときに用いる．これによりget upがきたとき、熟語としてのget upは辞書引きされる可能性があるが、動詞のget と前置詞のupが別々に辞書引きされることはなくなる．

```
v( [] , [] , [] ) --> [get,up] ,!.                    (1)
v( [] , [] , [] ) --> [get,on] .                      (2)
v( [] , [] , [] ) --> [get ] .                        (3)
v( [get ] , [ [form!v_en ] ] , [] ) --> [got ] .      (4)
v( [get ] , [ [form!v_ed ] ] , [] ) --> [got ] .      (5)
```

Fig.6.1 Dictionary for 'get' written in DCG

```
dicta(get,[[v,[[],[],[]]]],
          [[up,[],
               [!,[[v,[[],[],[]]]]],
           [on,[[v,[[],[],[]]]]]).

dicta(got,[[v,[[get],[[form!v_en]],[]]],
           [v,[[get],[[form!v_ed]],[]]]],
          []).
```

Fig.6.2 TRIE sturucture dictionary for 'get'

Fig.6.1 をトランスレータにより変換した結果をFig.6.2 に示す。ＤＣＧではニつあったgot に関する節が一つのdicta 述語にまとめられており、got の付加情報の部分が

```
[ [v, [ [get ] , [ [formiv_en ] ] , [] ] ] ,
  [v, [ [get ] , [ [formiv_ed ] ] , [] ] ] ]
```

となっていることに注意。

TRIE構造の辞書ではcomputer interface、get upのようなに単に単語を並べた熟語だけではなく、not only ~ but also ~ のように途中の~ の部分に非終端記号を含む熟語も登録できる。この熟語をＤＣＧで書き表すとFig.6.3 のようになる。

```
np([],A,[]) --> [not,only],np(NP1_A,_),
                [but,also],np(NP2_A,_),
                   {check(NP1_A,NP2_A,A)}.      (1)
adj([],[],[]) --> [not,only],adj(_,_),
                  [but,also],adj(_,_).          (2)
adv([],[],[]) --> [not,only],adv(_,_),
                  [but,also],adv(_,_).          (3)
```

Fig.6.3 Dictionary for 'not only ~ but also ~' written in DCG

not only ~ but also ~ にの~ の部分は、名詞句(np)、形容詞(adj) 、副詞(adv) になることができるので、３種類のＤＣＧ節が必要である。(1) の最後の {} で囲まれた部分は、この熟語に補強したPrologプログラムである。述語check は熟語の中の名詞句と熟語全体とで、人称・数の属性を一致させる為の述語である。この述語は、別にPrologで定義しておく必要がある。

TRIE構造に変換するトランスレータは図6.3 のＤＣＧ節を次の図6.5 のように変換する。

```
dicta(not,[],
        [[only,[],
          [[np,NP1_A,_],[],
           [but,[],
            [also,[],
             [[np,NP2_A,_],[],
              [(check(NP1_A,NP2_A,A)),[[np,[[],A,[]]
                                                ]]]]]]],
          [[adj,_,_],[],
           [but,[],
            [also,[],
             [[adj,_,_],[adj,[[],[],[]]]]]]],
          [[adv,_,_],[],
           [but,[],
            [also,[],
             [[adv,_,_],[adv,[[],[],[]]]]]]]]).
```

Fig.6.4 TRIE sturucture dictionary for 'not only ~ but also ~'

補強用のPrologプログラム {check(NP1_A,NP2_A,A)} は、(check(NP1_A,NP2_A,A)) に変換して、辞書に埋め込む。一方、非終端記号の部分np([NP1_A] , _)は、リスト [np, [NP1_A] , _] に変換して、辞書に埋め込む。辞書引きのときは、アトムの代わりにリストがあると、そこでgoal節を呼ぶ。この場合はgoal(np , [[NP1_A] ,_] ,X,Y) が呼ばれる。

7．辞書引きアルゴリズム

辞書引きの原則はマッチする一番長いものを最も優先させることである．そのため、TRIE構造の辞書を先頭から順にたどり、マッチするものを次々とスタックし、それが失敗するまで繰り返す（勿論、最初に失敗したら、それが語形変化形であるかどうかが、自動的に調べられる［安川 82］）．従って、スタックの先頭に最も長い辞書引き結果が積まれることになる．まず、熟語ではなく、単語の辞書引きを見てみる．例えば、John got a book.のgot である．辞書は先頭の単語ごとに節になっているので、got の辞書を持って来る．ここでは、これは、Fig.6.2 の下の節である．辞書を持って来たら、辞書引きはマッチする長いものが優先されるので、got a という熟語の可能性を調べる．Fig.7.1 を見れば分かるが、got の辞書は熟語が入る第３引数が［］になっていて、got が最初に来る熟語はないことが、分かる．しかし、第２引数のgot に関する付加情報の所に［get］があってget の辞書に対する参照の指示がある．そこで、get の辞書を持って来る．これは、Fig.6.2 の上の節である．get の辞書の第３引数の、熟語に関する情報を辿ると、get に続く単語としてonとupがあるが、a はどちらも一致しないので、get にa が続く熟語がないことが分かる．そこで、got が熟語ではなく、単語であるとする．got の辞書の第２引数の、単語got に関する情報、

```
[ [v, [ [get ] , [ [form!v_en ] ] , [] ] ] ,
  [v, [ [get ] , [ [form!v_ed ] ] , [] ] ] ]
```

を持って来る．この情報は、リストになっていて、二つつながっているが、最初に

```
[v, [ [get ] , [ [form!v_en ] ] , [] ] ]
```

という、got が過去分詞であるという情報を使う．しかし、John got a book.のgot は、過去形のため、文法を使って解析を進めると、いずれ失敗してバックトラックして来る．そこで、次の

```
[v, [ [get ] , [ [form!v_ed ] ] , [] ] ]
```

を使う．

次に、not only tall but also heavyを例にあげて熟語の辞書引きのアルゴリズムを説明する．まず、not が先頭に来る語の入った辞書を持って来る．これは、Fig.6.5 であり、第３引数の熟語の部分は、Fig.7.1 のような構造になっている．

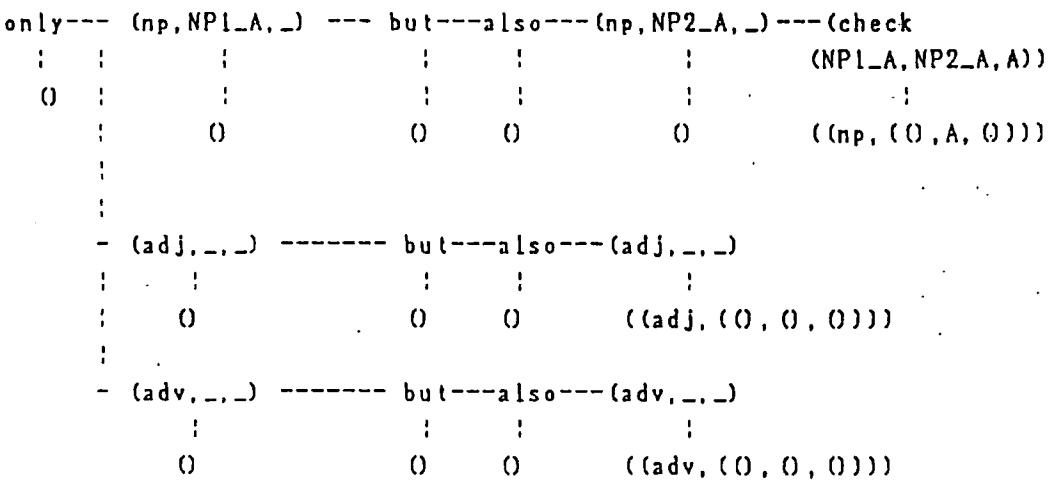

Fig.7.1 TRIE structure for 'not only ~ but also ~'

not only tall but also heavyは、not onlyまでは無条件にFig.7.1 とのマッチが進む．次の語に関しては、Fig.7.1 に示すように熟語辞書上で枝分かれして名詞句と形容

詞と副詞が来る．名詞句が先頭にあるので名詞句の可能性を試み、ここでは、goal(np,[[NP1_A],_],X,Y) が呼ばれる．まだ、tall but also heavy が残っているが、これでは、名詞句がないので失敗する．よって、形容詞を目指す．goal(adj,[_,_],X,Y) を呼ぶとtallが形容詞として成功する．そこで、真ん中の枝を辿ることになる．tallまでマッチし終わっているので、but とalsoをマッチさせると、これは成功する．再び形容詞が来るが、これはheavy が成功して、not only tall but also heavyが[[adj,[[],[],[]]]]だと分かる．

辞書引きは

dictionary(Cat,A,(not,only,john,but,also,mary,walks),X).

を呼ぶことによってなされる．Cat はカテゴリ名でnpが返される．A は辞書引きされた語の情報がリストになって返される．今の場合、

[[],[],[not,only,[np,[nomhd,john]],
but,also,[np,[nomhd,mary]]]]

が返される．最初の[] は、解析のためのアドバイス情報、２番目の[] は、意味情報、３番目はその語の構文木が入っている．第３引数と第４引数との差によってどの語が辞書引きされたが分かる．この例ではmaryまでが辞書引きに成功したので、第４引数は[walks] が返って来る．

8．ユーティリティ

そこで、ＤＣＧの形で書いた辞書をTRIE構造に変換するトランスレータを作成した．ＤＣＧによる記述の中で右辺の最初が終端項となっていて、ＯＲが使われていなければ、TRIEの辞書に変換できる．したがって、右辺の最初に非終端項、Prologプログラムがきてはならない．

TRIE構造の辞書は、プリティープリンタで打ち出しても、人が見た場合かなりわかりにくく、どの語が辞書に入っているか調べる場合など大変である．そこで、辞書を検索するためのプログラムを作成した．この検索プログラムでは各種のワイルドカードが使用可能なので、多方面からの辞書の検索が可能である．使用例をFig.8.1 に示す．

```
input words: not only + but also +.
Input category please: np.
not only (np,_373,_374) but also (np,_375,_376)
                          {check(_373,_375,_377)}
              Category: np
              Advice  : _377
              meaning : []
```

Fig.8.1 Retrieving dictionary

9．TRIE 構造の辞書による構文解析結果

TRIE構造の辞書を使って熟語を含む英文を解析した結果をFig.9.1 に示す．これらは文法規則４３７、辞書５８０語（内、熟語２０語）のもとで解析したものである．

解析時間をTRIE構造の辞書を用いたときと、従来の辞書を用いたときで比較した結果をFig.9.2 に載せる．文法、辞書の規模はほぼ同じである．ただし、従来の形の辞書を用いたときは、not only ~ but also ~ などの非終端記号を含む熟語は文法規則として扱ってある．

```
No. 1
 |-sentence
    |-sdec
       |-subj
       |   |-np
       |       |-not
       |       |-only
       |       |-np
       |       |   |-nomhd
       |       |       |-n -- john
       |       |-but
       |       |-also
       |       |-np
       |           |-nomhd
       |               |-n -- mary
       |-vp
           |-v
           |   |-v -- like
           |   |-suffix -- s
           |-obj
               |-np
                   |-pron -- him

number of wf_goal was :       6.
number of wf_dict was :       8.
number of fail_goal was :     39.
```

Fig.9.1 Sample parsing tree

解析文	解析時間（sec） TRIE構造	従来
This is a pen.	24.2	33.8
This is a pen that I bought yesterday.	88.0	98.5
The computer system works.	29.6	40.8
Not only John but also mary likes him.	31.2	44.2

Fig.9.2 Parsing time

熟語を含む時も含まないときも平均して2~ 3割程度,TRIE 構造の辞書が速くなっている．この理由として次の2つが考えられる．

第1は辞書がunit-clause で表わされるようになり、今まで右辺にあった単語が引数の中に入ったこと．つまり、従来の辞書は

```
dict(n, [] ) --> (john).
```

のような形だったので、語を引くとき、全てのdict節をたどらなければならなかった．しかし、TRIE構造の辞書では引かれる単語がunit-clause の引数の中にはいっているので、目的の節を一回で捜す事ができる．

第2に、辞書の節の数が減ったことをあげることができる．先に述べたように、従来は一つの熟語を表わすのに、一つの節を使っていたのに対して、TRIE構造では先頭が同

じ単語の熟語は全部、一つの節にまとめてある。また、dress のように、同じ単語が複数のカテゴリを持つ場合や、got のように過去形と過去分詞が同じ形の場合、従来はそれぞれに対して一つの節が必要であったが、TRIE構造の辞書ではカテゴリなどの付加情報をリストにして幾つでも付けられるようにしたので、節を一つにまとめることができる。ちなみに、辞書の節の数を比較すると従来の形の辞書が５６５、TRIE構造の辞書が５２０と、かなりTRIE構造の方が少なくなっている。辞書の中の熟語の数が多くなればこの差は更に広がるだろう。

１１．結論

辞書の中に非終端記号やPrologプログラムを埋め込むことにより、辞書の記述が柔軟で詳細になり、熟語も扱えるようになった。また、TRIE構造では熟語同士、単語が重複する部分はお互い共有することが出来るので、メモリが節約され、辞書引き時間が短縮された。この構造の辞書は見にくいことが欠点であったが、辞書検索の用のユーティリティを開発してこの問題を解決した。

熟語をＢＵＰで扱う試みは、ＥＴＬの松本らの方法［松本 84 ］が報告されている。これは、熟語をIdiom という節で定義し、辞書引きの時に辞書とは別に、これらを見るというものであった。これに対して、今回のTRIE構造の辞書の熟語処理は、熟語を完全に辞書と一体化できること、メモリが少なくてすむこと、トランスレータの使用により熟語を簡単に表記できること、単語の語尾変化の処理を自動的に行なうという特徴がある。

我々の方法による熟語処理に関して、残された問題は次の点である。

熟語の辞書引きが先頭の単語をたよりになされるので、最初に非終端記号が来る熟語は扱えない。つまり、

not only ~ but also ~

などの場合は、not が終端記号なのでそれをたよりに辞書を引けばよいが、

(所有格) own (his own,our own,..)

などの場合は、先頭に何が来るかわからないので、辞書をどのように引けばよいかわからない。しかし、このような熟語は、そうは多くないので文法規則として扱うことができる。

謝辞

御討論いただいた東京工業大学情報工学科田中研究室の諸氏に感謝します。

参考文献

[Pereira 80] :
F.Pereira & D.Warren,
"Definite Clause Grammar for Language Analysis
--A Survey of the Formalism and
a Comparison with Augmented Transition Networks",
Artificial Intelligence,13,pp231-278,1980

[安川82] :
安川秀樹、田中穂積、
Prologによる形態素処理と熟語処理
自然言語処理32-4, 1982.8.2

[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資料34-6, 1982

[松本 83] :
松本裕治、他、
"Prologに埋め込まれたボトムアップパーサ：ＢＵＰ"、
Proc. of THE LOGIC PROGRAMMING CONFERRENCE '83, 1983

[松本 84] :
松本裕治、浦野正樹
"構文解析システムＢＵＰの機能拡張"
Proc. of THE LOGIC PROGRAMMING CONFERRENCE '84, 1984

[C-Prolog manual] :
Pereira
"C-Prolog User's Manual Version 1.5"
February 22, 1984

[奥村 84] :
奥村学
"Prologによる日本語のボトムアップ構文解析"
1984 東京工業大学卒業論文

[田中 84] :
田中穂積、松本裕治
"自然言語処理におけるProlog"
「情報処理」 第25巻 第12号

[Aho 81] :
Alfred V.Aho,John E.Hopcroft, Jeffrey D.Ullman
"Data Structures And Algorithms"
ADDISON-WESLEY PUBLISHING COMPANY