自然言語処理 47－2
（1985. 1. 24）

文法的逸脱を考慮した柔軟な構文解析

井上 哲也　　上原 邦昭　　豊田 順一
（大阪大学基礎工学部）　（大阪大学産業科学研究所）

1．はじめに

人間が会話をする際、話し手が常に文法規則に従った言語表現のみを用いているとは限らない。それにもかかわらず、その文法的逸脱の度合いが少ない場合には、聞き手が柔軟に解釈・理解して、会話がスムーズに進行することが多い。こうした人間の行為を考えると、人間と計算機が自然言語でやりとりを行なおうとする場面に於いても、人間は、文法を逸脱した文を無意識的に入力することがあると考えられる。

文法的逸脱は人間(native speaker)が同意する逸脱か否かによって、絶対的逸脱と相対的逸脱に分類される。前者は、システムも人間も共に誤りと認める言語表現である。後者は、人間は誤りと思わないが、システムの用意している文法規則の限界を越えている言語表現である。それぞれについて、例を示す。

(1) 絶対的逸脱
- 文法的制約条件の誤り
  He want to buy these book.　（数）
  He shall lives in Tokyo.（動詞の活用）
  I saw he at the station.　（格）
  Give job description for Mike Smith.
  （冠詞の省略）
  Jack is is here.　（不要な語句の挿入）

(2) 相対的逸脱
- 擬人法や比喩による意味的制約条件の誤り
  The car drinks gasoline.
- 挿入語句
  Show me please the way to the station.
- 接続詞句
  John loves Mary and (John) hates Sue.

「接続詞句」は、接続詞句を受理する文法規則を用意している自然言語処理システムが少ないという認識の下に、相対的逸脱と分類している。

一方、従来の自然言語処理システムは各システムが個々に持つ文法規則に合致する入力文のみを処理対象とし、合致しない文が入力された場合には、単に拒絶して再入力を要求するのみであった。従って、実用性という面から見れば、不十分であったと考えられる。

以上のような認識に立ち、我々は、文法的逸脱を含む入力文に対しても柔軟に構文解析を行なうシステムＦＰＳ（Flexible Parsing System ）の開発を進めている[6]。ＦＰＳは、ＤＣＧモデル[8]を拡張し、接続詞句の解析処理・制約条件を逸脱した表現の柔軟な解析処理・文法規則に適合しない挿入語句の読み飛ばし処理の３つの機能を付加している。本稿ではＦＰＳの制御機構について述べる。

2．ＦＰＳの構文解析処理

図１にシステム全体の処理の流れを示す。以下、個々の処理について順に述べる。

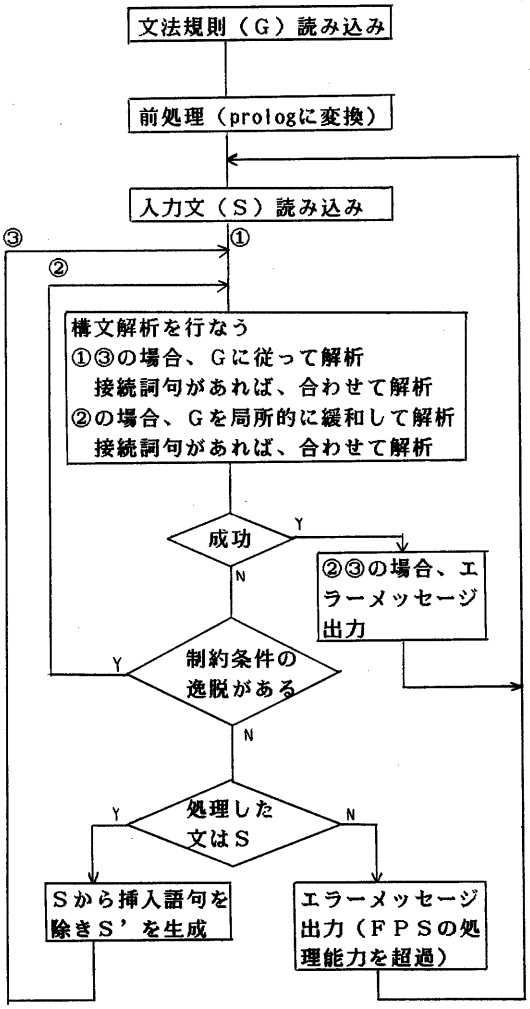

図１　ＦＰＳの処理の流れ

２．１　前処理

ＦＰＳの文法規則は、前処理によりprologの節に変換され、prologのプログラムとして評価される。(1) 接続詞句を処理するための書き換え規則、及び、(2) 文法的逸脱をチェックしている制約条件の内、緩和可能な述語の前処理は2.2 節と2.3 節で説明し、ここではそれら以外の文法規則の前処理について述べる。

①非終端記号の前処理

非終端記号には、入力単語列を保持するためのd_list（以下、X0,Xn と記述する）、入力文中に制約条件に反する表現があった場合に表示するエラーメッセージを保持するためのd_list（以下、E0,En と記述する）、及び、各文法規則に対して割り当てられた通し番号（以下、R_num と記述する）の５個の引数(extra argument) を付加する。ただし、R_num は接続詞句を解析するために再帰的に親ゴールを呼ぶ際、親ゴールを一意的に指定するために用いられる(2.2節で後述)。右辺中の非終端記号には５つ目の引数として無名変数(anonymous variable) を付加する。

②終端記号の前処理

X0,Xn の２引数を付加し、get_wordというＦＰＳの組み込み述語に変換する。get_wordはＤＣＧのconnects述語と同様の働きをする他、2.4 節で後述する「最長解析パスの管理」を行なう。中カッコ {} 内の述語（文法規則の適用条件を記述したもの。以下、テストと呼ぶ）は全く変換をうけない。

図２に示した(2-1) の文法規則はR_num=7 の場合、(2-2) のprologプログラムに変換される。

```
tovo([tovo,to,V,O],Core)
 -->[to],                                    (2-1)
    [V],{infinite_verb(V)},
    object(O,Core).

tovo([tovo,to,V,O],Core,X0,Xn,E0,En,7)
 :- get_word(X0,to,X1),
    get_word(X1,V,X2),                       (2-2)
    infinite_verb(V),
    object(O,Core,X2,Xn,E0,En,_).
```

図２　通常の文法規則の前処理例

２．２　接続詞句の解析

現在の自然言語処理技術では、接続詞を含む構文の解析は難解な問題の一つとされている。これは、接続詞の前後で共通に含まれる語句がある場合、繰り返しを避けるために共通部分の一方を省略した形で結合されるようなカテゴリもあるために、すべての接続詞句のパターンを文法規則で用意しようとすると文法が爆発してしまうという事情による。　まず、以下の説明で用いられる用語と記号を定義し、その後にＦＰＳの処理を説明する。

定義

カテゴリａとa'が <接続詞> によって接続されている場合を考える。

　　ａ <接続詞> ａ'　　(3-1)

このとき

　　ａ－＞ｂ，ｃ，ｄ．　　(3-2)
　　　①　②　③

また

　a'－＞b',c',d'.

とする。

ａもしくは『b,c,d 』の並びを「前連結句」(preconjunct) と呼ぶ。a'もしくは『b',c',d'』の並びを「後連結句」(postconjunct)と呼ぶ。『 <接続詞>,a'』もしくは『 <接続詞> ,b',c',d' 』の並びを「接続詞句」と呼ぶ。

ＦＰＳでは多種多様な接続詞句を、第１型接続詞句と第２型接続詞句の２種類のグループに分類し、それぞれのグループに対して、統一的な処理を施す。

２．２．１　第１型接続詞句

第１型接続詞句をとるカテゴリ（以下、「第１型カテゴリ」と呼ぶ）の代表的な例として文カテゴリ(sent)が上げられる。Sager の文法理論[9] に従えば、文の書き換え規則は一般に(3-3) のように書ける。

```
sent --> s,v,o,sa.                           (3-3)
```

s, v, o, sa はそれぞれ、主語、動詞、目的語（補語や二重目的語を含む）、副詞句等の文修飾語を表わす。

図３に示す例文１と例文２の接続を考える。ｓとs'(John)、及び、saとsa'(in the morning) が同一の語句であるため、これらの２つの文が接続されると、例文３、４、５のような種々の文が生成される。例文３は、ｏとsaの間に『 <接続詞>,v',o' 』の形の接続詞句が挿入されている例である。s'とsa' は、それぞれｓ及びsaと語句を共有して省略されている。例文４は、例文３と同様にｏとsaの間に接続詞句が挿入されている例であるが、s'(John)は明記され、sa' のみがsaと語句を共有して省略されている。このため接続詞句は『 <接続詞>,s',v',o'』の形で挿入されている。

例文１

John washes his face in the morning.
s v o sa

例文２

John reads the newspaper in the morning.
s' v' o' sa'

例文３

John washes his face and
s(s') v o

reads the newspaper in the morning.
v' o' sa(sa')

例文４

John washes his face and John
s v o s'

reads the newspaper in the morning.
v' o' sa(sa')

例文５

John washes his face in the morning and
John reads the newspaper in the morning.

図３　文カテゴリの接続例

例文５はsaの後ろに省略を全く含まない接続詞句『 <接続詞>,s',v',o',sa'』が挿入されている例である。

このように第１型接続詞句は文法規則右辺の各カテゴリの間に出現し、前連結句と後連結句の間で単一の語句を共有することがある。(3-2) の書き換え規則において、第１型接続詞句は①②③のいずれの位置にも出現する可能性がある。

①の位置に出現する場合
『 <接続詞>,b'』の形で出現し、『c',d' 』は『c,d 』と単一の語句を共有するため、省略される。

②の位置に出現する場合（例文３、４参照）
『 <接続詞>,(b'),(c') 』の形で出現し、d'はdと単一の語句を共有するため、省略される。(b') はbとb'が単一の語句を共有する場合、b'が省略され得ることを示す。(c')も同様である。

③の位置に出現する場合（例文５参照）
『 <接続詞>,(b'),(c'),(d')』の形で出現する。

第１型カテゴリの書き換え規則と前処理

第１型カテゴリの書き換え規則は、書式１に示すように、右辺の最後に組み込み述語conj1 を付け加えた形で記述される。

```
a --> b,c,d,conj1(C_word,a').      書式１
```

a'はa と同じカテゴリである。簡単のため、カテゴリの引数部は省略している。
組み込み述語conj1 は２引数を持つ。接続詞句が出現した場合には、接続詞(and,or,but,nor,",") など）が変数C_wordに返され、後連結句が第２引数のカテゴリa'によって解析されることを表わす。C_wordやa'を文法規則中に明に記述することによって、C_wordやa'の引数に返される接続詞句の解析結果を aの上位構造へ伝えることが可能となり、文法記述者は構文木や論理表現など任意の構造を組み立てることができる。 R_num=7 の場合、書式１に示した書き換え規則は図４に示す２つのprologプログラムに変換される。

```
a(X0,Xn,E0,En,7):-
    type1_ip([b,c,d],[7,C_word,a'],X0,Xn,
             E0,En,a(X0,Xn,E0,En,7)).
rule_base(7,a,[b,c,d],[7,C_word,a']).
```

図４　第１型カテゴリの前処理

a(X0,Xn,E0,En,7)は非終端記号a の引数部に2.1節の①で述べた５つの引数を付加したものである。type1_ipは右辺の各カテゴリをインタープリタ方式で逐一的に評価するＦＰＳの組み込み述語である。述語rule_base は、第１型カテゴリの書き換え規則をファクトの形で保持するものである。ＦＰＳは接続詞句を発見すると、 R_numとａをキーとしてrule_base を参照し、後連結句を解析するためのゴール列[b,c,d] を生成する。

第１型接続詞句の解析処理

右辺のカテゴリ列のどの部分に接続詞句が出現するか前もって決定できないため、ＦＰＳは各カテゴリを逐一的に評価し、接続詞句が出現すればそれをデモン的に処理する。接続詞句内部の解析においては、省略語句の扱いが問題となる。

以下、図３の例文３の解析を例にとり、組み込み述語type1_ipの処理を説明する。まず、文カテゴリの書き換え規則が書式１に従って図５の(5-1)のように書かれているとする。(5-1) は先に説明した前処理により(5-2),(5-3) のprologのプログラムに変換される。変数 SとNum はカテゴリsで、 Vはv で、0 はo で、Saはsaでそれぞれ値が定まるものとする。

type1_ipは以下の３つの情報を持つ。

(1)文法規則右辺のゴール列
[s(S,Num),v(V,Num),o(0),sa(Sa,0)]　(5-4)

(2)入力単語列を保持するためのd_list (X0,Xn)
エラーメッセージを保持するためのd_list (E0,En)
左辺のゴール(Head) [2.3 節で後述するように、右辺ゴール中に組み込み述語relax が含まれる場合、その引数として付加される]

```
sent([[sent,S,V,O,SA]|Tree1]) -->
  s(S,Num),
  v(V,Num),
  o(O),                                    (5-1)
  sa(SA,O),
  conj1(C_word,sent(Tree1)).

sent([[sent,S,V,O,SA]|Tree1],X0,Xn,E0,En,7)
  :-type1_ip([s(S,Num),v(V,Num),o(O),
      sa(Sa,O)],[7,C_word,sent(Tree1)],
      X0,Xn,E0,En,Head).                   (5-2)
      ただし、Head=(5-2)の左辺

rule_base(7,sent([[sent,S,V,O,SA]|Tree1]),
 [s(S,Num),v(V,Num),o(O),sa(SA,O)],
 [7,C_word,sent(Tree1)]).                  (5-3)
```

図５　文カテゴリの書き換え規則と前処理例

(3)文法規則(5-1) に割り当てられた番号 (7)
conj1 の引数　(C_word,sent(Tree1))
(5-4) のゴール列の処理中に接続詞句が出現する場合とそうでない場合がある。さらに、接続詞句が出現する場合にも、どのゴールとゴールの間に出現するか前もってわからない。従って、(5-4) のゴール列に、予め入力文やエラーメッセージの連続したd_listを割り当て、一連のprologプログラムとして実行することはできない。このため type1_ipは、(5-4) のゴール列から１つずつゴールを取り出して逐一的に評価し、接続詞が現われるとデモン的に接続詞句処理ルーチンを呼び出す。これは2.1 節の前処理で説明したように、取り出されたゴールに、非終端記号ならば[X0,X1,E0,E1,_] の５引数、終端記号ならば[X0,X1] の２引数、組み込み述語relax(3.3 節で後述) ならば[E0,E1,Head]の３引数をそれぞれ付加し、prologの述語として「コール」することで実現されている。例文３の場合、まずs(S,Num)が評価され、s(john,sing) となって成功する。続いてv(V,sing) と o(O)が評価され、それぞれv(washes,sing)、o('his face')となって成功する。

終端記号と非終端記号の評価が成功した場合、それらの評価結果

```
[s(john,sing),v(washes,sing),          (5-5)
  o('his face')]
```

を「評価済みゴール」と呼ぶ。

次に、接続詞and の出現により、sa(SA,'his face')の評価が失敗する。このように接続詞の出現により非終端記号、終端記号の評価が失敗した場合、もしくはゴール列が空になった場合、接続詞句が出現していないかどうか調べる。例文３の場合、(5-4) の残りのゴール列[sa(SA,'his face')] の評価を一時保留して、接続詞句処理に移る。

まず、出現した接続詞and を変数C_wordの値とする。次に、7(R_num)とsent(Tree1) の２つをキーとして(5-3) のrule_base を参照し(rule_baseの第１、第２引数とそれぞれパターンマッチする)、後連結句を解析するためのゴール列

```
[s'(S,Num),v'(V,Num),o'(O),sa'(SA,O)]  (5-6)
```

を生成する ((5-4) と区別するため、各カテゴリに 'を付けている)。

前連結句ではs, v, o まで解析が進んでいるため、後連結句では(5-6) のゴール列の内、o'まで順に解析を行なう。ところが、後連結句ではJohn が省略されているため、s'(S,Num) の評価が失敗してしまい、その後のv'やo'まで解析を進めることが出来ない。このため、s'(S,Num) と評価済みゴール(5-5) のs(john,sing)を比較し、s'(S,Num) の引数部のうちまだ変数である部分(SとNum)を、対応するs(john,sing)の引数部の値 (johnと sing) とパターンマッチさせる。この「省略語句の補足処理」により、s'(S,Num) はs'(john,sing) となって成功し、続いてv'(V,sing),o'(O)が評価される。これらは、それぞれv'(reads,sing)、o'('the newspaper') となって成功し、接続詞句の解析処理を終える。(5-6) で生成したゴール列の内、この段階でまだ評価されていない部分

```
[sa'(SA,'the newspaper')]              (5-7)
```

を「未評価ゴール」と呼ぶ。

先に一時保留していた(5-4) の残りのゴール列[sa(SA,'his face')] を再開する。sa(SA,'his face')はsa('in the morning','his face') となって成功する。この時、同一の語句を共有する未評価ゴールに対する補足処理を合わせて行なう。sa'(SA,'the newspaper') とsa('in the morning','his face') が比較され、sa' の変数SAの値として'in the morning'が代入される。
この段階で、(5-4) のゴール列が空になり、かつ新たな接続詞句もないため、例文３の解析が終了する。以上のようにして、接続詞句と省略語句の処理が行なわれる。

２．２．２　第２型接続詞句

第２型接続詞句をとるカテゴリ（以下、第２型カテゴリと呼ぶ）の代表的な例として、名詞や形容詞などを上げることができる。
図６に例を示す。

the easy homework and exams
名詞　名詞
a serve and devoted teacher
形容詞　形容詞
図６　第２型カテゴリの接続例

第２型接続詞句が出現するのは、(3-2) の③（右辺の最後尾）の位置に限られ、前連結句と後連結句の間で単一の語句を共有する場合はない。

第２型カテゴリの書き換え規則と前処理

第２型カテゴリの書き換え規則は、書式２に示すように、右辺の最後に組み込み述語conj2 を付け加えた形で記述される。（記号の意味は書式１と同様）

```
a --> b,c,d,conj2(C_word,a').
```
書式２

書式２に示した書き換え規則は図７に示すprolog プログラムに変換される。(R_num=7の場合)

```
a(X0,Xn,E0,En,7):-
    b(X0,X1,E0,E1,_),
    c(X1,X2,E1,E2,_),
    d(X2,X3,E2,E3,_),
    conj2(C_word,a',X3,Xn,E3,En,7).
```
図７　第２型カテゴリの前処理

3.1 節で示した接続詞句を取らないカテゴリの場合とほぼ同様の前処理がなされる。唯一の相違点は、右辺最後尾のconj2 に付加される５つ目の引数が、無名変数ではなく、R_num である点である。接続詞句が出現している場合には、R_num の番号を持つ文法規則（すなわち自分自身）を再帰的に呼んで後連結句を解析する。

第２型接続詞句の解析処理

接続詞句は組込み述語conj2 によって受理される。conj2 のプログラムを図８に示し、処理を説明する。

```
conj2(C_word,G,[C_word|X1],Xn,E0,En,R_num)
  :-member(C_word,[and,or,but,nor,',']),
    G=..[HIT],
    append(T,[X1,Xn,E0,En,R_num],T1),
    Goal=..[HIT1],
    Goal.
conj2([],G,X,X,E,E,_)
  :-nil_assign(G).
```
図８　conj2 のプログラム

①入力単語列の先頭語が接続詞であれば
・C_wordに接続詞を代入する。
・X0,Xn から接続詞をのぞいたd_list(X1,Xn) 、E0,En 及びR_num の５つの引数をG の引数部に付加してG を評価する。（図７の場合、再帰的にａの書き換え規則が呼ばれ、後連結句が解析される）
②入力単語列の先頭語が接続詞でなければ
C_wordとG の引数部の変数に[]を代入して終了する。

２．３　制約条件の緩和処理

制約条件の逸脱に対しては、それらをチェックしているテストを単に文法規則から取り除けばよいとする考え方がある。しかしながら、文法規則中に置かれる制約条件は、単に文法的整合性を調べているだけではなく、曖昧性を除去したり、誤った文法規則の適用を早期に排除することによって探索空間を制限し、解析効率の改善に役立っている場合が多い。したがって、単に文法規則から取り除くという方法は賢明な策とは言えない。

ＦＰＳでは、入力文に逸脱表現がある場合に限って、逸脱部分をチェックしているテストのみを局所的に緩和して解析を進めるという手法で対処する。

テストを緩和する際、次の２点に注意する。
(1) 文法記述者によって緩和可能であると指定されているテストのみを緩和の対象とする。緩和可能なテストは、書式３に示すように、組み込み述語relax の引数の形で記述される。

```
{relax(a,error_message)}
```
書式３

aはテストを表わす。第２引数には、入力文中にテスト aに反する表現がある場合に出力されるエラーメッセージが記述される。

組み込み述語relax には、前処理により、左辺の非終端記号とE0,En の３引数が付加される。左辺の非終端記号は、後述するように、「テストのマーク付け」に用いられる。例えば、図９の文法規則(9-1) は(9-2) のprologのプログラムに変換される。（R_num=7 の場合）

```
a(A1,A2) --> b(B1,B2),                    (9-1)
    {d(B1),relax(c(B2),error_c)}.
a(A1,A2,X0,Xn,E0,En,7):-
  b(B1,B2,X0,Xn,E0,E1,_),
  d(B1),                                  (9-2)
  relax(c(B2),error_c,a(A1,A2,X0,Xn,
        E0,En,7),E1,En).
```
図９　relax を含む文法規則の前処理例

(2) テストが失敗した場合でも、すぐ緩和するわけではなく、失敗したテストをいったんマーク付けしたうえでバックトラックして、他の解析パスで成功するものがないかどうか調べる。それも失敗し、先にマーク付けしたテストを緩和しなければ全く解析パスができないことを確認した後に緩和する。
次にアルゴリズムを示す。

①一度目の解析を行なう。
緩和可能と指定されているテストで失敗したときには、そのテストをマーク付けしておく。
成功した解析パスがあれば終了。
成功した解析パスがなければ②へ。
②前回の解析で新たにマーク付けされたテストが1つもなければ、「解析の失敗原因は制約条件の逸脱ではなく、文法規則に適合しない挿入語句にあった」と判断し、2.4 節で説明する「挿入語句の読み飛ばし処理」に移る。
新たにマーク付けされたテストが1つでもあれば、③へ。
③新たな解析を行なう。その際に
・マーク付けされているテストは緩和して解析を進める。
・まだマーク付けのされていない緩和可能なテストで失敗したときには、そのテストをマーク付けしておく。
成功した解析パスがあれば、緩和したテストのエラーメッセージを出力した上で終了。
成功した解析パスがなければ、②へ。

relax の引数の形で記述されているテスト t が親ゴール p の文法規則の右辺に埋め込まれているとする。この時、テスト t を「マーク付け」することは、t が評価されようとする時点の(t,p) のペアをテーブルに登録 [table(p,t)をassertする] ことによって実現される。

テストのマーク付け、テストの緩和、d_listによるエラーメッセージの管理等の処理は組み込み述語relax が行なう。relax のプログラムを図10に示す。

```
relax(Test,Message,Head,E0,En)
  :-table(Test,Head),!,                緩和
    append(E0,[Message],En).
relax(Test,_,_,E,E)
  :-Test.                          テストの評価
relax(Test,_,Head,_,_)
  :-not(Test),
    assertz(table(Test,Head)),マーク付け
    !,fail.                        した後、失敗
```

図10 relax のプログラム

2．4 挿入語句の読み飛ばし処理

制約条件の緩和を試みても解析が成功しない場合、失敗原因は文法規則に適合しない余分な語句が入力文中に挿入されているためと考えられる。

挿入語句に対しては、それらを受理する文法規則を追加すればよいという考え方がある。しかしながら、絶対的逸脱となる挿入語句を前もって予期するのは実際問題として不可能であり、万が一可能だとしても解析すること自体無意味であると考えられる。一方、相対的逸脱となる語句の挿入はある程度予測可能であるが、すべてを文法規則で用意しようとすると、無制限に文法規模が増大して解析効率が著しく低下してしまう。

我々は、文法の規模は質問応答や機械翻訳といった各システムの目的にあった大きさにとどめ、その許容範囲を越える挿入語句は単に読み飛ばすべきであると考える。

入力文中に挿入語句がある場合、最長の解析パスは次のいずれかの段階で途切れる。
(1)挿入語句の先頭の語を読み込んだ直後
(2)挿入語句の先頭より後ろにある語をいくつか読み込んだ直後
(2) は挿入語句の一部が誤った文法規則の適用によって一時的に受理される場合である。ただし、「5語を越えるような過度に長い語句が挿入されたり、3語を越える範囲にわたって挿入語句の一部が受理されてしまうのはまれである」というヒューリスティックを設ける。この5語、3語という値は、文献[3] の例文を検討した結果得られた実験値である。

例えば、文頭と文末の2箇所で 'if+ 文' の形の副詞節を受理する規則は用意しているが、'if any'という熟語の受理規則は持っていないシステムを想定する。このシステムで図11の例文6、例文7を解析する場合を考える。

例文6
What is if any Jan Nelson's college degree?
例文7
What is Jan Nelson's college degree if any?

What is any Jan Nelson's college degree? (11-1)
What is Jan Nelson's college degree? (11-2)
What is Jan Nelson's college degree if? (11-3)

(11-1)(11-2)(11-3)は例文6、7から一部の語句を取り除いた文である

図11 挿入語句の読み飛ばし例

例文６では、ifを読み込んだ直後にifの受理が失敗し、解析パスが途切れる。そしてany 以下の語が読み込まれることがない。一方、例文７では、'if any'のifが副詞節のifとして受理されてしまう。そしてany を読み込み、主語として受理しようとする段階で解析パスが途切れる。

ＦＰＳでは解析を進めてゆく際、組み込み述語get_wordが、その時点までの最長解析パスでは入力文中の何語目までが読み込まれたかを常に管理する。この「最長解析パスを途切れさせていた語」を、本稿では、「ブロック語(blocking word)」と呼ぶ。

ｎ語からなる入力文の解析が失敗し、ブロック語がｍ番目の語であったとする。

ＦＰＳは、まず(1) の場合を想定し、ｍ番目の語から連続するｋ語(1 =< k =< min{5,n-m+1})を元の文から除いた新しい入力文で順次解析する。例文６の場合、まずブロック語(if)を除いた文(11-1)を生成して解析を試みる。この文の解析もやはり失敗するので、次にブロック語から連続する２語(if any)を除いた文(11-2)の解析を試みる。例文６の場合はこの段階で成功し、「if anyを読み飛ばした」というメッセージを出力した後、解析処理を終える。

ブロック語から連続する５語もしくは文末の語までを読み飛ばしても解析が成功しなかった場合には、(2) の場合と考えられるのでブロック語からさかのぼって読み飛ばし処理を行なう。まず１語さかのぼり、先程と同様に、ｍ－１番目の語から連続する語句を元の文から除いた新しい入力文で順次解析する。例文７の場合、まずブロック語(any) の１語手前のifを除いた文(11-3)の解析を試みる。この文もやはり失敗するので、次にifから連続する２語(if any)を除いた文(11-2)の解析を試みる。例文７の場合はこの段階で成功し、「if anyを読み飛ばした」というメッセージを出力した後、解析処理を終える。ブロック語からのさかのぼりは、先に設けたヒューリスティックに基づき、最大で３語さかのぼる段階まで行なう。

３．関連した研究

近年、文法的逸脱を考慮した構文解析システムがいくつか提案されている。本節ではそれらとＦＰＳを比較し、ＦＰＳの特徴をさらに明確にする。

３．１　接続詞句以外の文法的逸脱について

HaysとMouradian[5]は、文法規則と入力文の「パターンマッチ基準」を緩和し、断片文や挿入語句を含む文を解析する。しかしながら、その手法は、システムの適用範囲を電子郵便や医療診断等の限られた分野に制限し、各分野に応じた緩和処理を行なうため、彼ら自身が認めているように、「自然言語一般に対する柔軟性はない」。

Charniak[1] は決定的に解析を進める構文解析システムParagramを提案している。Paragramは、各解析段階において、その段階で最も正しいと思われる解析結果（文法的に100%正しくなくても構わない）を決定的に選択し、以後の解析を進めてゆく。

KwasnyとSondheimer[7] は、ＡＴＮの枠組みで、「アーク上のテスト」と「カテゴリの階層関係」を緩和する手法を提案している。

ＦＰＳはＤＣＧを拡張した、非決定的で汎用的な構文解析システムである。また、文法規則の緩和はテストの緩和に限られ、文法カテゴリの階層関係や順序関係を緩和することはない。

３．２　接続詞句の処理について

SYSCONJ[11] はＡＴＮに組み込まれた接続詞句処理機能である。解析の途中で接続詞が発見されると、その時点の「解析状態(configuration)」Ａを一時保留し、接続詞の後ろ部分が解析可能な状態Ｂまで解析パスをさかのぼる。そしてＢを再開して、接続詞の後ろを解析をする。この解析は先に保留したＡと「マージ」可能な状態まで続けられ、その後Ａを再開する。しかしSYSCONJ」には、1)解析パスをどこまでさかのぼればよいか、またＡをいつ再開すればよいかが過度に非決定的であるために、組み合わせ的に爆発してしまう。このため解析効率が非常に悪い。2)Gapping のように接続詞句の中心に省略がある表現を解析できない等の欠点がある。

DahlとMcCord[2] は、ＦＰＳと同様に、述語論理の枠組みで接続詞句を処理する手法を提案している。しかしながら、1)接続詞句はModifier Structure Grammar(MSG)というシステムに特有な文法体系の中で扱われるため、文法記述者独自の構造を組み立てることができない。2)省略を含む接続詞句を解析できない。3)解析の途中で接続詞が現われると、再帰的に親ゴールを呼び、失敗した場合には、さらにそのまた親ゴールを呼ぶという処理を繰り返す。従って、すべての非終端記号に対して接続詞句に接続されているかどうか調べるために、爆発する可能性がある等の欠点がある。

Sedogbo [10]は、メタ文法による接続詞句を処理を提案している。対象言語はフランス語である。1)接続詞句内で省略が予測される文法カテゴリに対し、文法記述者がnil 規則を用意する必要がある。2)接続詞句の外でも(1) で用意したnil 規則が適用されるので曖昧性が増大する。3)解析結果は構文木出力に限られ、文法記述者が任意の構造を組み立てることはできない等の欠点がある。

ＦＰＳでは、1)解析中に接続詞が現われた場合、接続詞句を解析するためのゴール列を現在処理中の親ゴールを再帰的に呼ぶことによって生成する。しかも、その親ゴールは接続詞句を取り得ると文法記述者によって指定されているカテゴリに限定しているので、SISCONJ やDahl等のような爆発はない。2)文法記述者は任意の構造を組み立てることが出来る。3)第１型カテゴリの接続詞句中では、任意のカテゴリが省略可能である。従って、Gapping を含め、省略を含む種々の接続詞句が解析できる。4)接続詞句の解析はＦＰＳが自動的に行なうので、省略が予測される文法カテゴリに対し、文法記述者がnil 規則を用意する必要はない。

４．おわりに

本稿では、与えられた文法規則を逸脱する表現を含む入力文に対しても柔軟に処理を進める構文解析システムＦＰＳの制御機構について述べた。ＦＰＳは、ＤＣＧモデルを拡張し、1)接続詞句の解析、2)メタ述語relax による文法的制約条件等の局所的な緩和、3)文法規則に適合しない挿入語句の読み飛ばし処理の３つの機能を付加している。

今後の課題として、1)ミススペリングや未知語など辞書レベルの文法的逸脱に対して、校正したりエラーメッセージを表示する機能の追加、2)人間が誤りを犯しやすい度合いに応じて制約条件をレベル分けし、１つの文に対して複数個の制約条件が緩和可能な場合には、人間が誤りやすい制約条件ほど優先的に緩和するようなアルゴリズムの導入などを考えている。

参考文献

[1] Charniak,E.:A Parser with Something for Everyone, In King, Margaret, ed., Parsing Natural Languages. Academic Press, New York, New York

[2] Dahl V.and McCord M.C.:Treating Coordination in Logic Grammars, American Journal of Computational Linguistics,9 ,2,pp.69-91 ,1983.

[3] Eastman,C.M.and McLean,D.S.:On the Need for Parsing Ill-formed Input, American Journal of Computational Linguistics, 7,4, pp.257,1981.

[4] Eastman,C.M.and McLean,D.S.:A Query Corpus Containing Ill-formed Input,Technical Report 84-cse-9,Southern Methodist University, Dallas, Texas

[5] Hayes,P.J.and Mouradian,G.V.:Flexible Parsing, American Journal of Computational Linguistics, 7, 4, pp.232-241,1981.

[6] 井上，上原，豊田：文法的逸脱を考慮した柔軟な構文解析，情報処理学会第２９回全国大会予稿集 PP.1175-1176,1984.

[7] Kwasny,S.C.and Sondheimer,N.K.:Relaxation Techniques for Parsing Ill-formed Input ,American Journal of Computational Linguistics, 7, 2, pp.99-108,1981.

[8] Pereira,F.C.N.and Warren,D.H.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, 1980.

[9] Sager,N:Natural Language Information Processing, Adison-Wesley, Reading, 1981.

[10] Sedogbo,C.:A Meta Grammar for Handling Coordination in Logic Grammars, Proc. of Natural Language Understanding and Logic Programming, Rennes-France, pp.137-149, 1984.

[11] Woods,W.A.:An Experimental Parsing System for Transition Network Grammars, In Rustin,R., ed., Natural Language Processing, Algorithmics Press,New York, New York , 1973.