自然言語処理 72－2
（1989. 5. 19）

# HPSGに基づく英文の構文チェッカ

中野幹生　建石由佳　津田宏　小野芳彦　山田尚勇
東京大学理学部情報科学科

素性のミスマッチを検出することによって英文の構文チェックを行なうシステムを作製した。このシステムはHPSGパーザを修正したもので、いくつかの素性がミスマッチしても解析を続け、ミスマッチを引き起こした語句を特定する。あいまいな構造を持つ文に対しては、最も誤りの度合の少ないものを選び、それに基づいて診断を下す。従来の構文チェックの方法では、二度パーズしなくてはならなかったが、この方法では一回のパーズで構文誤りを発見できる。ただし、複数の辞書項目を持つ語が多いと計算時間がかかるなど実用化にはまだまだ困難があることがわかった。

# A Syntax Checker Based on HPSG

NAKANO Mikio, TATEISI Yuka,
TUDA Hirosi, ONO Yoshihiko, YAMADA Hisao
Department of Information Science, Faculty of Science,
University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, JAPAN
E-mail : c31400%tansei.cc.u-tokyo.junet@relay.cs.net

A syntax checker is constructed which finds syntax errors in English sentences by detecting feature mismatches. The checker, based on HPSG, continues to parse a sentence when it finds weak feature mismatches in the sentence, and reports words or phrases with the mismatch features after parsing the sentence. If the sentence is syntactically ambiguous, the checker chooses the parse tree that has the least serious grammatical errors, and diagnoses based on it. Conventional methods need parsing twice, with strictly grammatical rules and with relaxed rules, but this method needs parsing only once. A problem of computation time explosion still remains, when the input sentence contains several words which have many lexical rules.

## 1 はじめに

日本人の手によって書かれた英語の文には、文法的な誤りが含まれている場合が多い。ここで言う文法的な誤りとは、例えば次のようなものである。

(1) The number of cars have increased.

(2) We cannot discuss on this problem any more.

このようにスペルチェッカーにかからない誤りを発見するためには、与えられた文を完全にパーズすることが必要である。(1) にあるような誤りを発見するには、文の中で、どの語が主語で、どの語が主動詞であるかを知らなければならないので、表層情報からだけでは、チェックを行なうことはできない。また (2) にあるような誤りを発見するには、精密に構築された辞書が必要である。

自然言語パーザを用いた構文チェッカの設計はいくつか試みられているが、共通した問題点は、文法的でない文をいかにパーズするかということである。普通パーザは、文法的に正しい文のみを解析し得るものだからである。

古くから用いられてきた方法は余分な規則を増やしてパーズする方法である [Heidorn82, 河合 84]。厳密に文法的な規則の他に誤りを許す規則を増やし、もし前者のみでパーズ出来れば、誤りがないと判断し、後者の規則を用いなければパーズできないときは、誤りがあると判断する。そして増やした規則のうちどの規則を用いてパーズしたかによって誤りの種類がわかる。

しかしこの方法には欠点がある。それは、入力文を二回パーズしなくてはならないことと、規則を増やしたために時間がかかることである。例えば文法的に正しい文の規則

S → NP[三人称単数] VP[三人称単数]
S → NP[三人称複数] VP[三人称複数]

と、非文のための規則

S → NP[三人称複数] VP[三人称単数]
S → NP[三人称単数] VP[三人称複数]

は、非常に似ているので、これらを別々に適用するのは無駄である。

この無駄を回避する方法として、素性 (feature) を使う方法がある。素性を使ったユニフィケーション文法では、上の二つの正しい規則は一つの規則

S → NP[人称 =x, 数 =y] VP[人称 =x, 数 =y]

で書くことが出来る。素性が単一化するかどうかで、文法的かどうかを知ることが出来る。[Kudo88] は、LFG を用いて構文誤りのチェックおよび修正をする方法を提案している。この方法だと規則も少なく、効率的にチェックができる。しかし、いく通りにも解釈できる文については、どの解釈に基づいて誤りを指摘するかということについて触れていないので、曖昧文を多く含む文に適用するのは無理がある。

本稿では、同じくユニフィケーション文法である HPSG[Pollard88] を用いて曖昧性のある文をも含む現実の文の誤りを発見する方法について述べる。我々は、システムを設計するに当たり、次のことを重点目標とした。

- 妥当な時間で解析できること。
- 入力文の構造を正しくとらえられること。
- 広い範囲の文が解析できること。

実用的な構文チェッカを作るためには、多種類の誤りを見つけることよりも、これらの条件を満たす方が、大切である。HPSG はルールの数が少ないので、HPSG パーザは効率的に文をパーズすると予測できる。HPSG パーザが構文誤りの発見のためどのくらい有効であるかを確かめるために、我々は実験システムの作成を試みた。

## 2 誤りの検出機構

### 2.1 SUBCAT 素性原理の修正

HPSG では統語範疇はいくつかの素性を持っている。そして素性は構造化されており、素性値が素性構造であることも有り得る。SUBCAT(下位範疇化) 素性原理によれば、構文木の各ノードに対し、その主辞の SUBCAT 素性の値と補語の範疇 (素性構造) は単一化しなくてはならない。言い替えると、規則

$$M \rightarrow C_1\ C_2 \cdots H \cdots C_n$$

という規則が適用できるための条件は H の SUBCAT 素性が $C_1 \cdots C_n$ の素性構造と単一化することである。また修飾の規則に関しても同じことが言えるが、我々のアプローチは修飾については [Pollard88] と異なっている。これについては後述する。

SUBCAT 素性原理を厳密に適用するパーザは文法的に正しい文だけしかパーズすることができない。しかし、いくつかの素性を単一化の条件からはずすことによって、文法的でない文をパーズすることができる。単一化の条件からはずさない素性を強素性

$$\begin{bmatrix} phonology & runs \\ major & \begin{bmatrix} maj & v \\ subcat & \begin{bmatrix} major & \begin{bmatrix} maj & n \\ sem & X \end{bmatrix} \\ minor & \begin{bmatrix} person & 3rd \\ number & singular \\ case & subj \end{bmatrix} \end{bmatrix} \\ form & fin \\ sem & \begin{bmatrix} reln & run \\ runner & X \end{bmatrix} \end{bmatrix} \\ minor & [tense\ present] \end{bmatrix}$$

図 2: runs の範疇

$$\begin{bmatrix} phonology & they \\ major & \begin{bmatrix} maj & n \\ sem & they \end{bmatrix} \\ minor & \begin{bmatrix} person & 3rd \\ number & plural \\ case & subj \end{bmatrix} \end{bmatrix}$$

図 1: they の範疇

(major features) と呼び、単一化の条件からはずした素性を弱素性 (minor features) と呼ぶことにする。例として 'they runs' という文を考えてみよう。they と runs の統語範疇は図 1、図 2のような素性構造である。従来の SUBCAT 素性原理によれば、主辞の MAJOR | SUBCAT の値と補語の範疇が単一化しなければこの二つの範疇から木を作ることはできない。これは they の MINOR | NUMBER と runs の MAJOR | SUBCAT | MINOR | NUMBER が単一化しないので失敗する。しかし、我々の誤り発見のアプローチでは、木をつくる条件は主辞の MAJOR | SUBCAT | MAJOR と補語の MAJOR が単一化することなので、they と runs から they runs がつくられる。

ここで新しい素性 ERROR を導入する。ERROR の値は、どこでどの素性のミスマッチが起こったかを示す。すなわち、ミスマッチした素性の名にミスマッチをおこした両側の子の PHONOLOGY を組み合わせたものとした。they runs の ERROR は、NUMBER がミスマッチしたことを示す (図 3)。

$$\begin{bmatrix} phonology & they\ runs \\ major & \begin{bmatrix} maj & v \\ form & fin \\ sem & \begin{bmatrix} reln & run \\ runner & they \end{bmatrix} \end{bmatrix} \\ minor & [tense\ \ present] \\ error & [they(number)runs] \end{bmatrix}$$

図 3: they runs の範疇

ERROR の値は、子から親へと伝般される。つまり、あるノードの ERROR の値は子ノードの ERROR の値にそこのノードでミスマッチした素性の情報を加えたものである。従ってルートのノードの ERROR の値は、どこでどのような誤りが検出されたかを示す。これが即ち診断となる。

## 2.2 あいまいな文の誤り検出

入力文がいく通りにも解析できることはしばしばある。特に解析木をつくる条件に弱素性の単一化を含めないので、解析木は更に増える場合がある。このときどの解析木に基づいて診断を下すかということが問題となる。

> John bought a book about computers which are very popular.

この文では、which の先行詞は computers であるが、3 人称単数現在形の一致を無視した場合、a book about computers が先行詞であるとも考えられる。我々は

このように曖昧さを含む文を診断するために、次のような方法を用いた。

まず、誤りのない解析木が得られた場合は、解析をそこで終了し、誤りなしの診断を下す。誤りのある文については、ただ一つの解析木を選びそれに基づいて診断を出す。選ぶ解析木は誤りの度合が最も少ないものとする。なぜならば、誤りのもっとも少ないものが、書き手の意図した構造である可能性が強いと思われるからである。この誤りの度合を定量的に表わす指標として、コストを次のように定義する。

$$cost = \sum_{n} \sum_{k} w_k m_{k,n}$$

ここで$k$は、弱素性の番号、$n$は、ノードの番号、そして$w_k$は、$k$番目の素性ミスマッチから生じる誤りの重大性を示す重みである。また$m_{k,n}$は$n$番目のノードで素性が単一化した場合には0、失敗した場合には1となる。$w_k$の決め方については、後で述べる。このコストが最小であるような解析木を選び、それに基づいて診断を下す。例えば、

The strong will save I.

という文では、倒置法を許さなければ、

[$_{NP}$ The strong ] [$_{VP}$ will save I. ]
[$_{NP}$ The strong will ] [$_{VP}$ save I. ]

の2通りの解釈が可能である。NUMBER(数)の誤りの重みが2、CASE(格)の誤りの重みが1であるなら、最初の解釈のコストは1、2番目の解釈のコストは3となる。従って、最初の解釈に基づいてCASEの誤りの診断のみを出す。

## 3 文法

システムで用いた文法は、前に述べたようにHPSGを基にしたもので、主辞素性原理、束縛素性原理はそのまま用いた。

修飾に関して、HPSGでは修飾される側に情報が含まれている。すなわち、各範疇におけるADJUNCT素性の値はその範疇を修飾することができる範疇の集合である。しかし、これをそのまま使うことは、計算時間を増大させる。なぜなら、ADJUNCT規則が適用できるかどうかを知るためには、修飾句の範疇と、主辞のADJUNCTの要素1つ1つとを単一化させてみなければならないからである。そこで我々はJPSG[Gunji86]の定式化にしたがって、修飾する側にどのような範疇を修飾できるかという情報を書いておくことにする。

次に、HPSGでは、ID/LP形式を用いており、構成素の順序は独立した別のルールとなっているが、ボトムアップ解析をすすめる上で非常に不都合である。なぜなら、間違った順序の2つの句を結びつけようとして無駄な時間を費やしてしまうことがあるからである。

この無駄の回避のためにUNIFY素性を導入する。UNIFY素性はsubcatとadjunctを統合したもので位置情報を含めたもので、PERFORM, DIRECTION, CATEGORYの3つの素性の組のリストである(図4)。PERFORMはsubcatまたはadjunctの値をとり、DIRECTIONは、rightまたはleftの値をとる。また、CATEGORYの値は、1つの範疇である。PERFORMがsubcatでDIRECTIONがrightならば、右側にある句を補語として下位範疇化できる。ただし、その補語の範疇は、CATEGORYの値と単一化しなくてはならない。

各規則においてはUNIFYの先頭の要素だけを扱う。これによって規則は全て2分木だけとなり、規則の数も減りパーズは簡単になる。規則は基本的にM→CH, M→HC, M→AH, M→HAの4つである。ただしMは親、Hは主辞、Cは補語、Aは付加語である[1]。ルールの1つM→CHは詳しくは図5のようになる。この図で、インデックス[1]はHのMAJOR | UNIFY | CATEGORYと、Cの素性構造が単一化しなくてはならないことを示す。

## 4 インプリメンテーション

### 4.1 パーザ

パーザは、Prologで書き、VAX8600UNIX上のC-Prologで走らせた。アルゴリズムは、BUP [Matsumoto83] をHPSG用に修正したものに、誤りの検出機構とコストの計算機構を組み入れたものである。コストは解析と同時に計算する。即ち、規則を適用するごとに、弱素性がマッチするかどうかを調べてコストを足し上げていく。

複数の解析木が得られる入力に対しては、次の方法で解析を行なう。まず、最初の解析木を求める。もしも解析に失敗した場合、failという診断を出す。解析に成功し、コストが0の解、即ち文法的であると判断できる解が得られた場合、診断を下さずに、解析を終了する。コストが0でない解が得られた場合は、バックトラックして、コストの更に小さい解を探す。このとき解析途中で、コストがいままで得

[1] 実際のシステムでは、slashが導入されるので規則の数はいくらか増えるが本稿ではその説明は省略する。

$$\left[\begin{array}{ll} phon & runs \\ major|unify & \left\langle \left[\begin{array}{ll} perform & subcat \\ direction & right \\ category & [\ldots] \end{array}\right], \left[\begin{array}{ll} perform & subcat \\ direction & left \\ category & [\ldots] \end{array}\right] \right\rangle \end{array}\right]$$

図 4: UNIFY 素性

$$M[major|unify\ <rest>] \rightarrow$$

$$C\boxed{1}\ \ H\left[major|unify\ \left\langle \left[\begin{array}{ll} perform & subcat \\ direction & left \\ category & \boxed{1} \end{array}\right], <rest> \right\rangle\right]$$

図 5: ルールの例

られた解のコストと同じになったり超えたりした場合は fail して、さらにバックトラックする。解析過程においてコストが減ることはないからである。こうして最終的にコストが最小の解を得る。

## 4.2 後置修飾の扱い

長い文においては、後置修飾句がどこに係るのか判断するのが難しい時がある。また後置修飾句を含む文を文頭からパーズするとき、バックトラックが必ず必要である。

例えば、

I read the book that he wrote.

という文では、普通の left-corner 型パーザでは、I read the book までで 1 つの文と判断してしまい、その後に続く that he wrote を発見して、バックトラックする。the book と the book that he worte は、ほとんどその範疇を変えないので、このバックトラックは、無駄である。この不必要なバックトラックをなくすために、後置修飾は別の方法で扱う。後置修飾のルール M→HA を次のように書き換える。

- 後置修飾句 A に対しある H の右子孫 D があって、D→DA が成り立つとき、$M \overset{specialrule}{\rightarrow} HA$ が成り立つ。

ここで、右子孫は次のように定義される。

- M をルートとする subtree において、M は M の右子孫であり、また右子孫の右側の子も右子孫である。

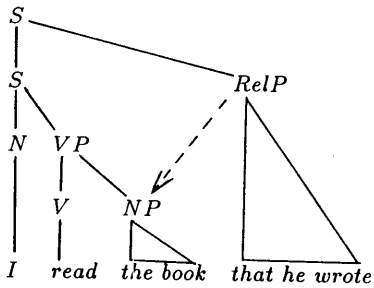


図 6: 後置修飾

この規則を用いることにより、後置修飾をバックトラックなしで扱える。例えば、先の例文は、I read the book までを解析しその後 that he wrote という後置修飾句を得る。that he wote は、I read the book の右子孫の一つ the book を修飾することができるので、I read the book that he worte を解析できる。これを図示すると、図 6のようになる。(この図では簡単のために範疇を伝統的な記号で略記した。)

## 4.3 強素性と弱素性

誤りの検出に使われ、単一化しなくても解析を続ける弱素性をどのように選ぶかがシステムの性能を大きく左右する。弱素性を増やせば、多くの種類の誤りを発見できるが、たくさんの過剰な構造を作ってしまい計算量が爆発することが有り得る。どの素

性を弱素性にし、どの素性を強素性にするかをアルゴリズミックに、または、定性的に決定する方法はまだ得られていない。われわれは、ある程度の基準を定めて決定した。まず、木を作るか作らないかの条件で、中心的な役割を果たす素性、即ち、MAJ(品詞), UNIFYを、強素性とすることはほとんど動かせない。次に、その条件を緩めることによって可能な構造の数が増大するであろうと予想できる素性については、強素性とした。例えば、VFORM(動詞の形)を弱素性にすると、the man listening to the radio などの分詞で修飾された句を the man listens to the radio の誤りであるとみなしてしまう。主動詞は、可能な構文木をしぼりこむのに重要な役割を果たしているので、その候補になる語を多くすることは得策でない。よってVFORMを強素性とした。その他の素性で、弱素性にしても計算時間があまり変わらないと思われるものは、全て弱素性とした。

## 4.4 発見できる誤り

次にあげる例は、弱素性を用いてどのような誤りが発見できるかを示す。文の後に書いた括弧のなかは、ミスマッチした素性である。

1. [The boys] [knows how to play baseball.] (NUMBER)
2. [Every] [girls] want to be brides. (NUMBER)
3. We [did not know] [when he comes here.] (TENSE)
4. He will come back in [a] [hour.] (VOWEL)
5. [Each] [the man] has his dream. (DETERMINER)
6. The book have been written [by] [Bob and I.] (CASE)

診断は、解析の結果に基づいて次のような形で出される。

```
[[each],determiner_disagree,[the,man]]
```

このように診断をみると、どこでどのような素性のミスマッチがあったのかを知ることが出来る。しかし、素性のミスマッチの原因がどの単語にあるのかをつきとめることまではできない。例えば上の例なら each の存在が悪いのか、the の存在が悪いのかは判らない。よって我々は、単に素性の種類と場所のみを示すことにした。

## 4.5 発見できない誤り

前の節ではどのような誤りが発見されるかについて述べたが、この節では発見不可能な誤りについて述べる。素性を用いた誤り発見では、構文的に正しい文と完全に構造の異なる文を解析することは不可能である。我々のアプローチでは、強素性のミスマッチを引き起こす様な誤りを持つ文を解析して診断を出すことはできない。例えば次のような文がそのような文である。

1. I can speaking Engilsh.
2. He turned pale immidiate.

1では、canのMAJOR | UNIFY | CATEGORY | MAJOR | VFORMとspeaking EnglishのMAJOR | VFORMがミスマッチするため、規則M→IICが適用できず、パーズに失敗する。このとき、何が原因でパーズができなかったかという情報を保持しておくことができないので、適切な診断はくだせない。

また2では、paleまでの解析には成功するが、imidiateが修飾する句がないので規則が適用できず、これも失敗する。この場合も、失敗の原因についての情報を保持せずに、バックトラックするので、パーザには、解析が成功しなかった理由はわからない。

我々のシステムは、このような誤りに対しては、単にfailという診断を出す。

## 4.6 結果と問題点

現在のバージョンのシステムでは強素性8つと弱素性13個を使用している。また辞書はそれほど精密ではないが、約4万語を持っている。いくつかの文を実験的に解析させた結果、次のような事がわかった。

1. 複数の辞書項目がある語(あいまい語)が多いと、計算時間が増大し、実用には耐えないこと。
2. あいまい語がなければ、多くの場合に適切な診断を出すこと。

成功した例を図7に示す。

# 5 実用化に向けて

## 5.1 自動詞と他動詞

我々のアプローチでは発見できない種類の誤りがいくつかある。先に述べたように強素性の単一化に

```
Sentence = [i,have,an,banana]
Cost = 1
Diagnosis = [[[an],voweldisagree,[banana]]]

Sentence = [the,old,green,bag,in,your,big,white,box,is,green,my,bag,
which,i,like]
Cost = 2
Diagnosis  = [[[green],detdisagree,[my,bag]]]

Sentence  = [a,old,man,like,a,girl,like,you]
Cost  =  4
Diagnosis  = [[[a],voweldisagree,[old,man]],
              [[a,old,man,like,a,girl],nomperdisagree,[like,you]]]

Sentence  = [i,eat,eat,cakes]
Cost = 10
Diagnosis  = fail
```

図 7: 出力結果

失敗するような誤りを含む文には、failという診断しか出せないためである。例えば、自動詞と他動詞の混同は日本人のよく行なう誤りであるが、自動詞と他動詞は、UNIFYの値のみによって区別できるので、我々のシステムでは適切な診断を出すことができない。

## 5.2 重みの決定

次に、重み$w_k$の値は現在では、我々の勘に頼っているのだが、より正確な診断を出すために、次のような方法で、求める計画を立てている。

あるノードで$i$番目の弱素性がミスマッチを起こすような誤りを書き手がする確率を$p_i$とする。すると、$m_{n,k}$が1となる確率は$p_k$となる。また、構文木が書き手の意図と関係なく作られた場合、$i$番目の弱素性のミスマッチが起こる確率$q_i$をとする。$cost_{rnd}$がランダムに作られた構文木のコストで、$cost_{int}$は筆者の意図した木のコストとする。われわれは、

$$P = Pr(cost_{int} < cost_{rnd})$$

が最大になるように$w_k$を選ぶ。$p_i$,$q_i$に関して2次までの近似でPを計算すると、

$$P = N\sum_i q_i - \frac{N(N+2)}{2}\sum_i\sum_j q_iq_j$$
$$+N^2\sum_i\sum_j p_iq_jh_{ij} - N^2\sum_i\sum_j p_iq_j$$
$$h_{ij} = \begin{cases} 1, & w_i < w_j; \\ 0, & otherwise, \end{cases}$$

となる。但し、$N$はノードの数である。$P$は$w_k$の大きさの順序のみに依存する。第3項が最大になるように$w_k$の大きさを決定すればよい。そのためには、$p_i$, $q_i$ を求めなければならないが、これらは多くの文を実験的に解析させることによって得られる。

## 5.3 計算速度

システムの実用化に向けて超えなければならない最大の障害は計算時間の問題である。現在のシステムでは、先に述べたようにあいまい語があると非常に多くの時間がかかってしまう。ある程度以上のあいまい語があると計算が爆発してしまうことが判った。この問題を解決するためには、バックトラックを起こりにくくするようなバージングアルゴリズムか、または、ヒューリスティックスを用いて可能性のない組み合せを削除する方法が必要である。

もしあいまい語がなければ、非常に短かい時間で解析できる。たとえば、図7の2番目の文でも1秒以内で解析できる。したがって、もし、可能性のない組み合わせを解析の初めの段階で削除できれば、十分に実用化が可能である。

## 6 終りに

HPSG パーザを用いて英文の構文誤りをチェックする機構を示した。素性のミスマッチを容認して解析を進めることにより文法的でない文も扱うことが可能となる。また曖昧な文に対しても適切な診断を出す方法を示した。最後に、計算時間の短縮が実用化に向けての大きな研究課題であることを述べた。

## 参考文献


[Heidorn82] Heidorn, G. E., K. Jensen, L. A., Miller, R. J. Byrd, and M. S. Chodrow, "The EPISTLE Text-critiquing System," *IBM Systems Journal 21(3)* pp. 305-326 (1982).

[Kudo88] Kudo, I., H. Koshino, M. Chung, and T. Morimoto, Schema Method: A Framework for Correcting Grammatically Ill-formed Input, *In Proceedings of 12th International Conference on Computational Linguistics* :341-347 (1988).

[Pollard88] Pollard, C. J., and I. A. Sag, *Information-Based Syntax and Semantics, Vol. 1: Fundamentals,* CSLI Lecture Notes Series No. 12, Center for the Study of Language and Information, Stanford University (1988).

[Matsumoto83] Matsumoto, Yuji, Hozumi Tanaka, Hideki Hirakawa, Hideo Miyoshi, and Hideki Yasukawa. BUP: a bottom up parser embedded in Prolog. *New Generation Computing, 1(2)*: 145-158 (1983).

[河合 84] 河合敦夫, 杉原厚吉, 杉江昇, 英文の誤りを検出するシステム ASPEC-I, 情報処理学会論文誌, Vol.25, No.6, pp. 1072-1073 (1984).

[Gunji86] Gunji, Takao *Japnese Phrase Structure Grammar.* Dordrecht: Reidel (1986).