単一化文法に基づく
自然言語解析用オブジェクトクラス群「言の葉」

松　原　康　夫

"Kotonoha" a set of object classes for natural language analysis based on unification grammar.

Yasuo Matsubara

Natural language understanding is one of the most important problems in artificial intelligence. Until now, the research has been done mainly for English statements. Research for Japanese statements also has been done, but it has not become to the level of one for English. Moreover, it seems that grammatical theory of Japanese has not been proposed in a form easy to analyze by computer.

In these days the main interest of research is about meanings of statements. However, it does not mean that the part of analyzing the statement syntactically and taking out the meaning has less importance.

A number of grammars called unification grammar which incorporates a mechanism of unification, have been proposed and indicated to be useful for analysis of syntactical phenomena. It is suggested that unification grammars are easy to analyze by computer and are useful to take out the meaning. In many applications of the area of artificial intelligence, the man-machine interface has great importance. If one can interact with an application through a language similar to the natural one, it can be used by unskilled people.

In this paper, we propose a set of C++ classes which can be used to incorporate a language analyzing function into any application program.
When a dictionary and a grammar are given in a fixed manner, these can analyze an statement and output a list similar to F-structure of LFG.

1．はじめに

自然言語をコンピュータで処理する研究には大きい二つの流れがある。一つは異なる言語間で翻訳することを目的とするものであり、もう一つは自然言語で書かれた文の意味をコンピュータに理解させようとするものである。これらはほぼ同じ年代に米国で研究が始まっている。特に翻訳は一時その可能性が否定されたこともあったが、現在では特定の言語間や、ある限定された内容については一応の成功を収め、実用に供されている。自然言語理解はそれ自身、人工知能の中

でも最も重要な分野の一つであり、いろいろな分野のプログラムで自然言語、またはそれに近い人工言語をマンマシンインターフェースとして採用するものも多い。しかしながら、本格的な自然言語の文章をコンピュータに理解させることは、極く限定された場合を除いて成功していない。

自然言語理解の研究[9]は、他の人工知能の研究と同様に米国が主導権を握っている。これに対して、我が国に於ける自然言語理解の研究は、未だ米国と同じ水準に達したとは言い難い。その原因は幾つか考えられる。日本語は使う文字の種類が非常に多いこと、分かち書きを行なわないことである。さらにもう一つ重要なことは、コンピュータの処理に向くような形での日本語文法が整えられていない事が考えられる。これらの問題のうち、文字種類が多いことについては、仮名漢字変換やローマ字漢字変換の普及によって解決されたといってよいであろう。これは一つにはハード的に多種類の文字が表示できるようになったこともあるが、自然言語処理の技術をソフトに応用したものと考えられる。

一方分かち書きを行なわないことによる困難[1)2)]の克服は、未だ解決されるべき問題点の一つである。これに対するいくつかのアプローチは存在するが、どれか一つが決め手としては普及してはいない。

ところで、コンピュータで自然言語文を解析するためには何らかの文法理論が必要となる。これまでの多くの研究では文脈自由文法が使われている。しかしこれだけでは文法規則の数が大きくなることや、非常に長い文等への対応が難しいことが指摘されている。また、文脈自由文法に基づく構文解析結果から意味を取り出す方法については必ずしも整理されているとは言い難い。そして句構造文法は元来、英語またはそれに近い言語の解析を目的として発達してきたため、日本語の解析には必ずしも向いていない可能性がある。

これに対して近年、単一化と呼ぶ操作[1)3)4)5)7)8)10)]を行なう機構を組み込んだ、単一化文法(Unification Grammar)と呼ばれる一群の文法理論が注目されつつある。これらの文法理論は、Chomsky等の変形文法に対する批判からでてきたものであるが、同時にコンピュータによる処理にも適した文法理論であるといえる。代表的なものとして、GPSG、LFG、CUG等があるが、従来の文脈自由文法の雰囲気を良く残しているのはLFGであろう。LFGでは構文を表わすC構造とその他の属性を表わすF構造に分けて考えており、まず文脈自由文法に基づいて構文解析を行なってC構造を作り上げた後で、単一化操作によって文全体のF構造を作り上げる。このため、従来良く研究されてきた構文解析の技法の利点を活かす事ができるものである。そして、F構造の中から意味を取り出すことも比較的容易である。

自然言語、中でも日本語に関する処理の研究は、文法理論の整備、意味理解の研究や他のアプリケーションプログラムのマンマシンインターフェースなど、多方面から研究する必要がある。その際、文脈自由文法を自然な形で拡張したと考えられる単一化文法に基づくことが、今後の極めて有力な方向である。これまで従来の文脈自由文法に基づいて文を解析するシステムに関しては、極めて多くの研究がなされているが、単一化文法に基づいて文を解析するシステムについては未だ限られた研究しかなされていない模様である。

一般に、自然言語処理を行なうプログラムはかなり大きいものになり、しかも研究の進展に伴って多くの変更を行なうものと考えられる。このような場合にはオブジェクト指向プログラミングの考え方が有効となる。特に他のプログラムのマンマシンインターフェース部として使う時は、言語処理部分は他の部分とは独立した部分として動作することになる。また、意味理解システムにおいても、単一化文法に基づく処理と、意味に関する処理とはある程度独立したものとなる。これらのことを考え合わせると、単一化文法に基づく文解析システムを、オブジェクト指向プログラミング言語のいくつかのクラスの集合として実現することが望ましいことになる。

現在、オブジェクト指向言語は多くのものが提案されているが、最も広く使われているのが

C++である。この言語はこれまで広く普及してきたC言語を拡張したものである。最近では、処理系の普及度と処理速度等の点から、従来はLISPやPROLOG等のいわゆる人工知能用言語(リスト処理言語)によって書かれていたプログラムを、CやC++等に書き替えることが行なわれている。

本論文では、特に日本語の解析を単一化文法に基づいて行なうシステムを、C++のクラスによって実現したシステム「言の葉」について述べる。このシステムではLFGと同様に、句構造規則に単一化のためのルールを付加したものを文法として読み込み、単語に品詞といくつかの属性を付加したものを辞書として読み込む。また構文解析にはボトムアップ並列型のアルゴリズムであるCKY法に、候補単語をすべて登録する方法を組み合わせた方法を用いる。このことによって、分かち書きしないことによる問題を克服している。

2．自然言語処理の流れ

一般に自然言語の処理は、初めに形態素解析を行ない、次に構文解析、そして意味解析を行なう。さらにこの後、意味や文脈に対応する処理を行ない、今度は逆に意味から、それを表現する文を生成する。但し本論文で対象とするのは、最初の形態素解析から構文解析、そして意味解析の入り口のところまでである。もちろんこの後の処理を行なわなければ完全な意味での自然言語処理とはいえない。しかしこの部分はどのような処理でも必要とする共通な部分である。この部分について一般的に使えるツールが存在するならば、その後の意味処理等の研究を促進するものとなる。

2．1　形態素解析

形態素解析は、入力された文を辞書上の見出し語、またはその変化したものに分解することである。英語などのように単語毎に分かち書きされているものについては、各単語毎にその品詞と変化形を定めることが主な仕事となる。これに対して日本語の場合はどこが単語の切れ目であるかを決めることが重要な仕事の一つとなる。このとき、形態素解析だけで100%正しい切り方を定めることは不可能である。そこで発見法によっていくつかの可能な切り方のリストを出力するのである。発見法としては最長一致法や、文字種の変わり目を目安にするもの、あるいは分節数最小法などがある。

辞書にどの段階の語を載せるかは重要な決定事項である。つまり、基本形と可能な変化の種類を載せるか、またはすべての変化形を見出し語として載せるかである。規則的な変化に対しては基本形から導くことが出来るが、不規則な変化をするものについては全ての変化形を見出し語として辞書に載せる必要がある。

さらに形態素解析として重要な仕事は、辞書に無い語、いわゆる未知語の処理である。自然言語で使う語を予めすべて辞書に載せておくことは、対象となるエリアをかなり限定しない限り困難である。そのため、常に新しい語が現れる可能性が存在する。それでも正しい語として処理しなければならないが、日本語の場合はどこからどこまでが未知語なのかを特定する必要がある。これも絶対的な方法はなく、文字種類等を手掛かりとする必要がある。

2．2　構文解析

形態素解析で提案された語の列に対して、ある文法に従って構文の木を見いだす。文法として文脈自由文法を使う事が多く、Chomskyの変形文法等を用いる事も研究されたが一般的ではない。文脈自由文法を使った構文解析は極めてよく研究されており、種々のアルゴリズムが存在す

る。

これらを分類する基準の一つは、トップダウン型かボトムアップ型かということである。トップダウン型では文記号Sから開始し、文の方に向かって規則を適用して行く。ボトムアップ型では文の方から、これを生成する規則を見いだして行く。これらをミックスしたものもある。また、直列型か並列型かの分類もある。直列型では可能な一つの選択肢を可能な限り深く追究するものであり、行き詰まったときにバックトラックを行なう。並列型では、適用可能な選択肢をすべて保持しながら処理を行なう。それぞれ一長一短があるが、最近はコンピュータの記憶容量が大きく安価であることを考えると、並列型の方が以前より有利であるかもしれない。とくに直列型の場合は下位部分に於ける同一の試行錯誤を何回も繰り返す可能性がある。

また、形態素解析で可能な切り方が複数あるとき、並列型ならばこれらを同時に並行して処理することも可能である。

2．3　単一化と意味解析

文脈自由文法は、自然言語の性質をかなりよく表わしているのでよく用いられるが、これに従って解析した構文木から、どのようにして意味を取り出すのかという方法論は、必ずしも整理されているとは言い難い。また文脈自由文法だけでは意味的に関連のある文を統一的に扱えなかったり、規則の数が多いなどの問題も存在する。例えば肯定文と疑問文は意味的に関連がありながら、文法的に独立な規則で処理されることになる。

Chomskyはこうした問題を文法の枠内で扱うために、変形規則を使ったがこれについては問題点も多い。こういった問題に対してChomskyとは異なる方法を取ったのが単一化文法である。これは記号に自由な数の属性を持たせ、これらの間の関係を、句構造規則に付随した単一化操作によって規定するものである。GPSG, CUG, LFGなどが提案されているが、従来の文脈自由文法の雰囲気を最もよく残しているのがLFGであると考えられる。LFGでは構文を表わすC構造と、それに付随する属性の構造であるF構造を区別している。最初にC構造を求め、その後にF構造を求めるので、最初の段階で従来の文脈自由文法に関する構文解析の技法を十分に活用することができる。また、F構造のなかに意味を表わす構造を含ませることで、文の意味を自然な形で取り出すことができる。

本論文では、LFGに近い形での、単一化文法の記述を可能とするシステムを構築する。

2．4　単一化操作と構文解析の並行動作

前節で述べたように、LFGでは先にC構造を構築してからF構造を構築している。しかし、C構造だけの解析では多くの可能性が残ってしまい、並列な構文解析法では全ての可能性を試すので、記憶容量と処理時間を消費することになる。理想的には意味の処理をも並列に行なうことによって、多くの選択肢をカットすべきであるが、意味の処理をある程度は含む単一化の処理を、構文解析と並行して行なうことによって、かなりの選択肢をカットできる可能性がある。

構文解析の操作とF構造の単一化操作を並行して行なう場合、以下のような点について注意が必要となる。F構造については、下位の記号に付随するF構造から上位の記号に付随するF構造を構成するのである。仮にトップダウンの構文解析アルゴリズムを使った場合、上位の部分木のF構造は下位の部分木に依存して決定される。しかし、初めからどの下位部分木が正しいものであるかは不明なので、上位の部分木のF構造については下位の部分木の選択肢の各々に対応して、別々のF構造を保持しなければならなくなる。

これに対して、ボトムアップ型の構文解析アルゴリズムの方が、単一化処理と並行して行なうのに適している。但し、単一化の操作によって下位の部分木のF構造を変化させてしまう場合に

は同様の問題を生じる。

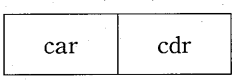
以上の考察に基づき、本論文ではボトムアップ並列型の構文解析アルゴリズムであるCKY法と、単一化操作によって下位のF構造を変化させないような性質を持つ単一化文法とを組み合わせ、構文解析と単一化の操作を同時に並行して行なう方法を提案する。また併せて、分かち書きの問題をも同時に解決する。

3．リスト処理のためのデータ構造

自然言語を始めとする、人工知能上の記号を主体とする処理においては、一般にリスト構造を多用する。このデータ構造をどのようなものにするかということはその表記法とあいまって、単に処理の仕方に影響を与えるというだけではなく、ものの考え方にも影響を与える一つの文化のようなものである。これはプログラムの構築に大きな影響を与え、最終的に実現性や効率を左右するものである。これまではLISP言語の影響が大きく、同様のデータ構造を使うことが多い。しかし本論文では、やや異なるデータ構造を採用しており、その表記法とともに予め記述しておく。

3．1　LISPのデータ構造

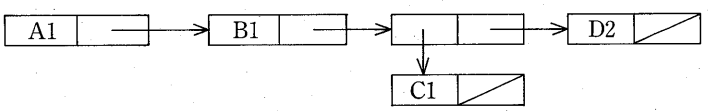
LISPにおいては以下のような二つの要素からなるセルをビルディングブロックとして使っている。



左側の部分はcar（カー）と呼び、右側の部分はcdr（クダー）と呼ぶ。一般にcar及びcdrには他のセルへのポインタを入れる。例えば次のようなリスト表記があった場合、

（A1　B1　（C1）　D2）

これは、セルをポインタで繋ぎあわせて次のような構造を意味するものである。



但し、ここでセルのcar部分に“A1”等の文字列が直接書いてあるのは、表記の便法であり、実際にはこのような文字列または直接数値がおいてある記憶エリアへのポインタが置かれる。このような文字列や直接数値の事をアトムといってそのための記憶エリアにまとめて置かれる。処理系はそのポインタがこのエリアを指していることによってアトムであると判断する。

このようにLISPにおいては具体的なデータ構造と、その表記方法が整っているため、同様のディシップリンがコンピュータサイエンスの中でも広く使われ一つの文化を形作っている。

3．2　本論文で採用するデータ構造

最近ではこれまでLISPやprolog等のいわゆる人工知能用の言語で書かれたプログラムを、C言語やC++言語等で書き替えることもよくある模様である。LISPやprologでは、リスト構造に関する処理が容易に書けたり、あるいはバックトラック等の処理が表現出来るという利点がある。しかし処理系自体が大きかったり、処理速度が遅い等の問題点も指摘されている。さらに大きいプログラムを書く際に全体をモジュール化して見通しのよいプログラムとするための工夫などにつ

いては、これまでの人工知能用言語は必ずしも優れているとはいい難い。

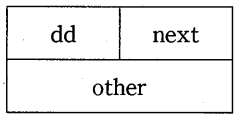
本論文では、自然言語処理に要求される機能が多種であり、研究の発展段階や応用の対象に応じて種々の変更が要求されること、また速度やプログラムのコンパクト性等をも考慮して、オブジェクト指向言語であるC++を使っての実現を想定している。

そのためには、必ずしもこれまで一般的であったLISPの習慣に捉われる必要はない。ここではとくに辞書構造やF－構造を表現するために、C及びC++上でリスト構造を実現するためのセルと、それによるリスト構造の表記方法を提案する。

具体的なセルとして次のようなC言語に於ける構造体を用いる。

```
struct dcell  {int dd; struct dcell  * other,  * next ; } ;
```

そしてこれは次のようなセルとして描くものとする。



このように、dcellにおいては具体的な値を入れる部分とポインタを入れる部分をはっきり区別している。これにより、LISPディシプリンにおいてアトムかどうかの判別が必要であった点を簡略化している。

次に述べておく必要があるのは、アトムの表現である。実際の処理においてはいろいろな文字列が出現する。具体的には各種の非終端記号や、F－構造の素性標識や素性値である。これらの文字列はその文字列そのものが問題なのではなく、互いに識別や同定が可能であれば良いのである。そこで本論文ではこれらの文字列を一つの整数に置き換え、これをdcellのdd部分に格納する方法をとる。これによって、互いの識別同定を、単なる整数値の比較によって行なう。

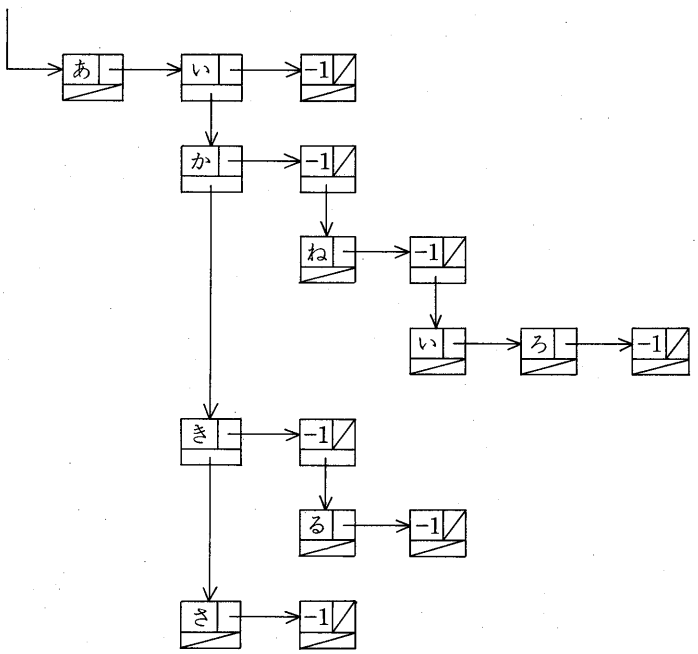


図3．1　trie構造のdcellによる構成

3．3　辞書のデータ構造

形態素解析の段階では、辞書を頻繁にアクセスすることになる。とくに、ある文字から探索を開始して引き続く文字を辞書の上で辿って行くような操作を可能とする必要がある。例えば「あいうえお」という文字列があり、この「あ」から開始する単語を辞書の上で探して行く。「あ」が辞書に載っていれば、その単語を候補単語として返す。次に「あい」が載っていればそれを候補単語として返す……というような操作が必要である。

このような操作に便利なのがトライ（trie）と呼ばれるデータ構造である。

例えば次のような一群の言葉が辞書に登録されているならば

「あき、あい、あきる、あさ、あか、あかね、あかねいろ」

これを表現するトライ構造は、上記のdcellを使って図3．1のように表現される。

ここで  はNULLポインタを表わし、何も指し示されないことを意味するのはLISPの場合と同様である。また、「あ」等の文字はこれに対応する整数値を意味する。つまり、引き続く文字に対してはnextポインタで繋ぎ、他の選択肢についてはotherポインタで繋ぐ。-1はそこで終了する単語が存在することを示し、実際はこの後に品詞やF-構造が繋がれる。

このようなtrie構造が使えるのは、単語の数が比較的小さい間のみである。登録単語数が大きくなった場合は、これを索引ファイル等で置き換える必要がある。そのようなデータ構造の置き換えは、辞書を一つのオブジェクトとして構成しておけば、他の部分に影響が及ばないため比較的容易である。

3．4　Ｆ－リストのデータ構造と表記

LFGにおいては、各非終端記号が持つ属性をＦ－構造によって表現している。Ｆ－構造はいくつかの素性標識と素性値の対の集合である。そして素性標識の種類と、各標識について取り得る素性値の種類が定まっている。

しかし本論文では、もっと素性標識や素性値に自由度を持たせ、さらにその対の並ぶ順序にも、時には意味を持たせたい。そこで本論文ではＦ－リストと呼ぶデータ構造を用いる。

Ｆ－リストは、標識と値の対からなる列である。空列のこともある。標識としては任意の、文字列を許す。値としてはアトムの場合とＦ－リストの場合がある。Ｆ－構造のように素性値によって取り得る値が制限されることはない。

Ｆ－リストは表示するとき左側の角括弧'['で始まり、右の角括弧']'で終わる。但し、'['と']'は別の行の同じカラム位置に表示し、その内容はこれらより右側に表示する。

各標識の値は、アトムかＦ－リストか、または空であることを許す。値の終わりには、それが、空であってもアトムであっても、Ｆ－リストであってもその後に';'セミコロンを置く。また、このＦ－リストをdcellで表わす時は、アトムの前に-1をdd部の値とするセルを置いて表わす。

例えば次のようなＦ－リストは

```
[ A1   [ B1  c ;
         B2   [ BB  b1 ;
              ] ;
       ] ;
  A2    B2 ;
  D1   [  E1     eee;
          E2     ;
       ] ;
] ;
```

dcellを使って図３．２のようなデータ構造を表わす。

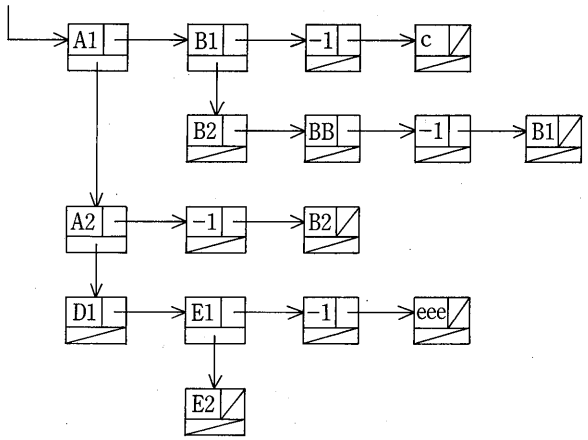


図３．２　Ｆ－リストのデータ構造

ここで“A1”等の文字列は、これに対応する整数がここに入っていることを意味する。

３．５　記憶管理

一般にリスト処理においては、メモリの管理が問題となる。新しいセルを造り出すために、未使用記憶エリアから一定の大きさのエリアを割り付けるのである。問題は一方で新しいセルが必要となるのに対して、他方では不要となるセルが生ずる事である。LISPでは新しいセルが必要となって、未使用エリアが無いときには、ガベージコレクション（ごみ集め）と呼ぶ操作を行なう。これは現在使用中のリスト構造に組み込まれているセルに何らかの印をつけていって、最後に印のついていないセルを未使用セルとして登録する方法である。

しかしごみ集めは時間が掛かり、処理を途中で中断して行なうので、できれば避けたい方法である。これを避けるためには、新しくリストを造るときには、これまでのリストの一部を流用せず必ず新しくセルを使って造りなおす方法がある。こうすれば不要になったリストに含まれるセルを全て不要のものとして未使用リストに戻すことができる。しかしこれでは余分な記憶容量を必要とするし、内容をコピーするための時間も必要とする。

そこで以下のような方法が現実的な妥協案として考えられる。つまり、処理の途中では従来と同じように、他のリストの一部であっても新しいリストに組み込む事を許す。しかしあるまとまった処理が終わって、最終結果を造るときには未使用セルのブロックを切り替えて、新しく内容をコピーし直すのである。こうすることで、これまで使っていたセルがある一定のアドレス範囲に納まっていれば、ここは改めて新しい未使用エリアとして使えることになるのである。とくにLISP等のようにごみ集めの機能がシステムに組み込まれていない、CやC++等の言語を使ってシステムを構築するときには有効である。

４．形態素解析と構文解析の融合

先述したように、分かち書きを行なわない日本語においては、入力文を単語に分解するやり方が複数存在する。そのため、従来いくつかの発見法を使って、出来るだけ少ない数の切り方を候補として挙げ、それらを一つ一つ構文解析する方法がとられた。しかしながら、本論文ではいろいろな応用を想定しており、とくに対象領域を限定した、疑似日本語等の処理も考える必要があ

る。そうした場合、必ずしも従来の発見法に基づく切り方が有効とは限らない。むしろその疑似日本語に対応した発見法を見つけなければならず、現実的ではない。

この問題を解決する方法として、本論文では可能な単語の候補を全て三角行列に登録し、構文解析の段階でボトムアップでしかも全ての可能性を並列に追究して行くアルゴリズムであるCKY法を適用する方法を提案する。これまでに採用された切り方の発見法も本来構文上許される接続関係だけを取り出す方法と考える事ができる。それは文法による制約に優先するものではなく、文法によって制約されるものである。したがって、本論文で提案する方法を使えば、辞書に載っている単語の範囲ならば、改めて入力文を単語に分解するための発見法を考える余地は無い。

4．2　形態素解析

本論文では、形態素解析は辞書引きを行なうことで実現する。つまり、辞書内部でどのような処理を行なうかに関わらず、ある文字の綴りが辞書に載っているかどうかを、辞書に問い合わせることで、形態素解析を行なう。

このような機能を持った辞書を、一つのオブジェクト“jisho”として構成し、辞書引きは、このオブジェクトのメソッドを呼び出す事で行なう。

C++では、各オブジェクトのクラスに対して必ずコンストラクターと呼ばれるメソッドが存在し、新しいインスタンス（そのクラスの具体的なオブジェクト）が造られたときに必ずこれが実行される。jishoオブジェクトの場合は、コンストラクターの中で辞書データを含むファイルを読み込み、トライ構造を構築するものである。

重要なメソッドとしてinit(), step(), item()が用意してある。init()は引き数をとらず関数値も返さない。ある文字から、短い順に辞書引きを行なう前に一回だけ実行することによって、オブジェクト内部のポインタをトライ構造のルートを指すようにするものである。step()は一つの漢字のコードを表わす整数を引き数として、トライを一文字分辿るものである。もしこの一文字が辿れなければ0を返し、ここで終わる文字があれば−1を返す。それ以外は1を返すものである。item()は辞書に登録された単語に関する記載項目を一つずつ取り出すものである。具体的には記載項目を表わすF−リストへのポインタを返す。

これらの機能を使って辞書引きを行なう手順は以下の様になる。

使う変数：

int　str[]；解析すべき入力文を入れた配列。配列要素には一つの文字のコードが入る。文の終わりは0で示す。

int　i,　j；配列strの要素を指す添え字。

int　s；関数stepの関数値を入れる変数。

手順：

①i=0；として先頭の文字を指す。

②str［i］が0であれば⑩に行く。

③init()；を実行する。

④j=i；とする。

⑤str［j］が0であれば、i=i+1；を行なって②に行く。

⑥s=step(str［j］)；現在見ている文字を引き数としてstepを呼び出す。

⑦もしs==0ならば、これ以上長い文字列は登録されていないので、

　　i=i+1；を行なって②に行く。

⑧もしs==−1ならば、str［i］からstr［j］までの文字列が辞書に登録してあるので、必要な処置をする。

⑨ j=j+1；を行なって⑤に行く。
⑩ 手続きを終了する。

例えば、辞書に「ある、はれ、はれた、ひ」という単語だけが登録されているとする。このとき「あるはれたひ」という入力文については次の様に辞書引きをする。

まずinit()を実行し、次にstep（あ）を行なう（実際には「あ」という文字のコードを整数型の変数に入れて呼ぶ)。結果は1が返ってくる。次にstep（る）を行なうが、これは辞書に載っているので-1が返ってくる。そしてstep(は)を行なうと0が返ってくるので、これ以上長い文字列の辞書引きを停止する。

またinit()を実行して、step（る）を実行すると0が返ってくる。

init()を実行して、step（は）を実行すると1が返ってくる。次にstep（れ）を実行すると-1が返ってくる。さらにstep（た）を実行すると-1が返ってくる。step（ひ）は0が返ってくる。

init()を実行し、step（れ）を実行して0が返る。

init()を実行し、step（た）は0を返す。

init()を実行して、step（ひ）は-1を返す。

このようにしてstep()が-1を返したときは、item()を使って記載項目を取り出す。

4．3　三角行列とCKY法

これまでのCKY法では、文が含む単語の数をWとして、W×Wの三角行列を使用した。例えば「わたしはここにいる」という文が「わたし　は　ここ　に　いる」と区切られてそれぞれの品詞がN, J, N, J, Vであれば、図4．1のような三角行列の一番下の行にその非終端記号を登録することから開始していた。

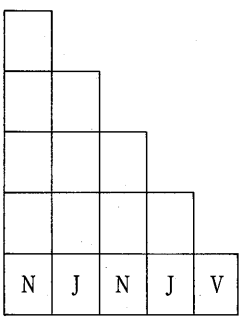


図4．1　従来のCKY法での三角行列

これに対して、本論文では入力文から見いだすことの出来る、可能な単語の候補を全て三角行列に登録する方法をとる。CKY法では、三角行列上の位置は、そこに登録された非終端記号の支配する範囲を表わしている。三角行列を配列tt [] [] で表わすものとして、tt [i] [j] に登録された非終端記号は tt [i] [0] から tt [i+j] [0] までの範囲を支配する（但し配列の添え字は0から始まるものとする)。逆に言えばこの非終端記号から、この部分を導出することが出来るのである。三角行列を図示するときは、図4．2の様にiを横方向、jを縦方向に変化させる。

tt [0] [4]				
tt [0] [3]	tt [1] [3]			
tt [0] [2]	tt [1] [2]	tt [2] [2]		
tt [0] [1]	tt [1] [1]	tt [2] [1]	tt [3] [1]	
tt [0] [0]	tt [1] [0]	tt [2] [0]	tt [3] [0]	tt [4] [0]

図４．２　サイズ５の三角行列と要素

つまり三角行列上の高さが、その支配する範囲の広さを表わすのである。そこで本論文ではこの高さが、支配する文字列の長さを表わすものと考える。したがって入力文に含まれる文字数が三角行列のサイズを決定する。

例として文「あなたはここにいる」という文を考える。但し、辞書には次のような単語が登録されているものとする。

N->あな，N->あなた，N->なた，J->は，N->はこ，DN->ここ，J->に，N->い，V->いる

この文は９個の文字を含むので、サイズが９の三角行列を使用し、上記の単語を登録すると図４．３のようになる。

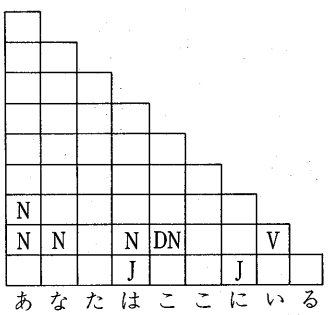


図４．３　三角行列への登録

一般的には、一つの行列要素に複数の非終端記号が登録されることがある。そして登録と同時に、シングルルールに関する処理を行なうことは従来のCKY法と同じである。

４．４　単一化規則を付け加えた句構造規則

CKY法では、文法がシングルルールかまたはダブルルールのいずれかであることを前提とする。句構造規則の右辺に丁度一つの非終端記号を持つものがシングルルールであり、２つあるのがダブルルールである。どのような文脈自由文法もこのような形に変形できることはよく知られた事実である。

本論文では、こうした書き替え規則に、単一化規則を付け加えたものをファイルから読み込み、Ｆ－リストの形で貯えるオブジェクトbunpouを用意している。

具体的な句構造規則は例えば次のように書いて、ファイルとして用意する。

```
1) S -> NP VP
[  X1 = X3;
   X1.SUBJ = X2;
];
2) NP -> N
[  X1 = X2;
];
  .  .  .
```

図４．４　単一化規則を付け加えた句構造規則の例

このように、句構造規則のあとに [] で括っていくつかの単一化規則を書く。ここでX1，X2，X3はそれぞれ左辺、右辺の左側、右辺の右側の記号のＦ－リストを意味する。

単一化規則としては、現在のところ以上のような等式のみをサポートしているが、将来必要に応じて機能を拡張する予定である。

等式の左辺には以上のようなXnまたは、Xnのあとにドット'.'でいくつかの文字列を繋いだものを許す。右辺には左辺と同様のものが書ける他、一つの文字列または二つ以上の文字列を'.'で繋いだものが書ける。

単一化の操作は、基本的にX2やX3のＦ－リストを変更しない。左辺がX1であるときは、右辺の情報を元にして、X1の構造に付け加えて行くことになる。X2やX3の内容を変えないので基本的に内容をコピーする。

X2，X3や等式の右辺で指定された標識が無いときは、単一化は成功しない。しかし左辺に現れたX1については、指定された標識を造って行く。

また同じ標識が異なる値を持っている場合も単一化は成功しない。一方の値が空になっていて、他方が値を持っているときは成功する。

４．５　CKY法と単一化の並行処理

CKY法のアルゴリズムはよく知られているので、ここでは述べない。ただ、本論文では構文解析をして行くのと同時に単一化の処理を行ない、また同時に木の構造を構成して行く。そのためのデータ構造として次のようなtcellを用いている。

```
struct tcell  {struct dcell  * dp;  int sw;
               struct tcell  * next,  * left,  * right;
              };
```

一つのtcellが丁度一つの非終端記号に対応する。一つの三角行列要素にいくつかのtcellを入れるので、これらを線形リストとしてつなぐのにポインタnextを使う。同様にポインタleftとrightは、木の構造でそれぞれ左側と右側の子を指すのに使う。非終端記号そのものや、それに付随するＦ－リスト構造などは、dcellを使って構成したリスト構造で表わし、それをtcellのdpにつなぐ。

したがって三角行列は、次のように大きめの配列として用意される。

```
struct tcell  * tt[100][100] ;
```

5．単一化文法と意味解析

単一化文法においては、文脈自由文法の各記号にいろいろな属性を持たせるのであるが、この中に意味に関する情報を含ませる事ができる。

本章では具体的な例によって、入力文から意味をどのようにして取り出すことができるかを示す。例として、生物に関するisa関係を定義する文と、逆にそれを質問する文を許す質問応答システムを示す。

5．1　対象領域

ここでは図5．1に示す範囲のisa関係を入力文によって作り上げ、また質問するシステムを想定する。

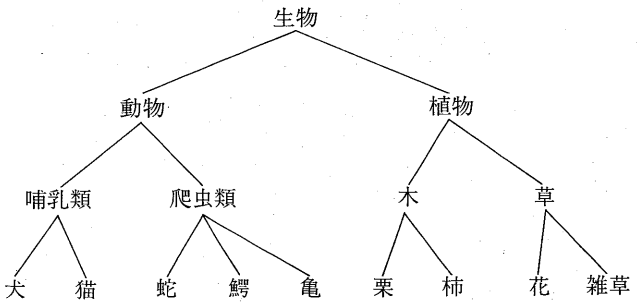


図5．1　生物に関するisa関係

そして例えば「犬は哺乳類である」という断定文や「栗は植物ですか」というような疑問文を許すことにする。断定文は直接的なisa関係を表わすものと解釈し、それまでにisa関係の無かったところにisa関係を設定する。但し、既に同じisa関係が設定されている場合はそれに対して確認の言葉を表示し、既に異なるisa関係が設定されている場合はエラーメッセージを出す。

疑問文の場合は直接的なisa関係だけでなく、間接的なisa関係に対しても、成り立つかどうかを判断して「はい」または「いいえ」で応答する。

以上のような応答をさせるために、一つの小さな世界を設定し、その中に存在するものを一つの線形リストに接続するものとする。この世界には、最初から図5．1に現れるものは存在するが、その間のisa関係が定義されていないものとする。一つのものはその名前と、isa関係の上位のものへのポインタによって表現される。

こういったリスト構造は、先に定義したdcellによって簡単に実現できる。つまりこの世界に存在するものを全て接続するためにnextポインタを使用し、isa関係を表わすのにotherポインタを使う。ものの名前は整数値に変換してdd部分に格納する。

5．2　文法と辞書項目

ここでは、あくまで単一化文法の処理から、意味を取り出す過程を描くことを目的としているので、各々の細かい属性は無視することにする。また、図5．1に登場する「もの」については、予め辞書に登録しておくものとする。

辞書を図5．2に、文法を図5．3に示す。

```
生物　N [ SEM seibutu;] ;
動物　N [ SEM doubutu;] ;
植物　N [ SEM shokubutu;] ;
哺乳類 N [ SEM honyuurui;] ;
爬虫類 N [ SEM hachuurui;] ; ] ;
犬 N [ SEM inu;] ;
猫 N [ SEM neko;] ;
蛇 N [ SEM hebi;] ;
鰐 N [ SEM wani;] ;
亀 N [ SEM kame;] ;
木 N [ SEM ki;] ;
草 N [ SEM kusa;] ;
柿 N [ SEM kaki;] ;
栗 N [ SEM kuri;] ;
花 N [ SEM hana;] ;
雑草　N [ SEM zassou;] ;
は J [ SEM ha;] ;
である V [ SEM  [dantei; OBJ1 ;  OBJ2 ;
                ] ;
         ] ;
です　V [ SEM  [dantei; OBJ1 ;  OBJ2 ;
                ] ;
         ] ;
ですか V [ SEM  [gimon; OBJ1 ;  OBJ2 ;
                ] ;
         ] ;
の J [ SEM no;] ;
上位概念 N [ SEM  [JouiGainen; SUB;
                ] ;
           ] ;
```

図５．２　Ｆ－リストを伴う辞書項目

```
1)  S -> NP1 VP
[ X1 = X3 ;
  X1.OBJ1 = X2 ;
] ;
2) NP1 -> NP J
[ X3.SEM = ha;
  X1 = X2 ;
] ;
3) VP -> NP V
[ X1.SEM = X3.SEM;
  X1.OBJ2 = X2 ;
] ;
4) NP -> N
[ X1 = X2 ;
] ;
5) NP -> NP3 N
[ X1.SEM = X3.SEM;
  X1.SUB = X2 ;
] ;
6) NP3 -> N J
[ X3.SEM = no;
  X1 = X2 ;
] ;
```

図５．３　単一化規則を付加した句構造規則

5．3　対話

以上のような辞書と文法に基づいて、実際に対話を行なった結果を示す。最初に各入力文に対してどのようなF－リストが出力されるかを示す。

例えば「花は草です」という文に対して、図5．4のように、文記号とそれに付随するF－リストが出力される。

```
[S [SEM  [dantei;
          OBJ1 ;
          OBJ2 ;
         ] ;
   OBJ2   [SEM kusa;
          ] ;
   OBJ1   [SEM hana;
          ] ;
   ] ;
] ;
```

図5．4　「花は草です」に対する出力

SEM以下の部分は「OBJ1はOBJ2であることを断定する」という意味である。
また、「植物は亀の上位概念ですか」という文に対しては図5．5のように出力される。

```
[S  [SEM  [gimon;
           OBJ1 ;
           OBJ2 ;
          ] ;
     OBJ2  [SEM  [JouiGainen;
                  SUB;
                 ] ;
            SUB  [SEM kame;
                  ] ;
           ] ;
     OBJ1  [SEM shokubutsu;
           ] ;
   ] ;
 ] ;
```

図5．5「植物は亀の上位概念ですか」に対する出力

この図において、SEM以下の部分は「OBJ1はOBJ2であるか」という意味であるが、OBJ2のSEM部分を見ることでOBJ2が「亀の上位概念」を表わすことが判る。

以上のような出力に対して、これを入力として応答するプログラムを一つのオブジェクトとして構成する。このオブジェクトはコンストラクタの他にkaiwa()というメソッドをもつ。このメソッドを上記のF－リストを引き数として呼び出す。このメソッドは、断定文に対して内部のisa関係をリンクとして設定し、疑問文に対してリンク関係から、その関係が成り立つかどうかを判断する。実際の会話の記録を図5．6に示す。

以上に示したように、辞書と文法によって各入力文に対する適切なF－リストを構成せしめ、そのF－リストから意味を取り出すプログラムを作成することにより、比較的容易に対話システムを構築することができる。

1)----------
入力文：花は生物ですか
応答文：その関係は成り立ちません
2)----------
入力文：花は草です
応答文：わかりました
3)----------
入力文：草は植物です
応答文：わかりました
4)----------
入力文：植物は生物です
応答文：わかりました
5)----------
入力文：木は植物です
応答文：わかりました
6)----------
入力文：雑草は草です
応答文：わかりました
7)----------
入力文：草は生物ですか
応答文：そうです
8)----------
入力文：柿は生物ですか
応答文：その関係は成り立ちません
9)----------
入力文：柿は木です
応答文：わかりました
10)----------
入力文：柿は植物ですか
応答文：そうです
11)----------
入力文：柿は生物ですか
応答文：そうです
12)----------
入力文：動物は生物です
応答文：わかりました
13)----------
入力文：哺乳類は動物です
応答文：わかりました
14)----------
入力文：爬虫類は動物です
応答文：わかりました
15)----------
入力文：猫は哺乳類です
応答文：わかりました
16)----------
入力文：犬は哺乳類である
応答文：わかりました
17)----------
入力文：鰐は爬虫類である
応答文：わかりました
18)----------
入力文：亀は爬虫類です
応答文：わかりました
19)----------
入力文：蛇は爬虫類です
応答文：わかりました
20)----------
入力文：蛇は動物ですか
応答文：そうです
21)----------
入力文：栗は生物ですか
応答文：その関係は成り立ちません
22)----------
入力文：栗は木ですか
応答文：その関係は成り立ちません
23)----------
入力文：栗は木です
応答文：わかりました
24)----------
入力文：栗は生物ですか
応答文：そうです
25)----------
入力文：end

図５．６

６．言の葉の表現能力

前章においては意味を取り出す過程に主眼を置いて説明したので、その他の文法的な事項は省略した。しかし「言の葉」はLFGにかなり近い表現能力をもっている。本章ではLFGと比較することによって、言の葉の文法的な表現能力を示す。

６．１　LFGとの比較

LFGでは例えば図６．１のような文法規則を書くことが出来る[1)]。

```
1)  S   ->    NP              VP
              (↑SUBJ)=↓       ↑=↓
2)  VP  ->    V               (NP)          (NP)
              ↑=↓             (↑OBJ)=↓      (↑OBJ2)=↓
3)  VP  ->    V     (NP)                    (PP)
                    (↑OBJ)=↓                (↑(↓PCASE))=↓
```

図6．1　LFGの文法規則

1)については言の葉でも一つの規則として書くことができる。図6．2の1)がほぼこれに対応するものと考えられる。

2)については、右辺の記号の数と省略記号があるので、複数の規則に分けて書く必要がある。しかしそれ以外は極めて容易である。図6．2では2)～6)で表現できる。

```
1) S     -> NP  VP [  X1.SUBJ  =  X2 ; X1  =  X3 ; ] ;
2) VP    -> V   [ X1  =  X2 ; ] ;
3) VP    -> V  NP  [ X1  =  X2 ;  X1.OBJ  =  X3 ; ] ;
4) VP    -> V  NP  [ X1  =  X2 ;  X1.OBJ2  =  X3 ; ] ;
5) VP    -> V  TEMP [ X1  =  X2 ;  X1.OBJ  =  X3.OBJ; X1.OBJ2  =  X3.OBJ2 ; ] ;
6) TEMP  -> NP  NP [ X1.OBJ  =  X2 ;  X1.OBJ2  =  X3 ; ] ;
```

図6．2　言の葉の文法規則

図6．1の3)については、右辺のPPについた(↑(↓PCASE))=↓という式が問題となる。これは文献(1)によればvarを変数として(↑var)=↓と(↓PCASE)=varの二つの式をまとめたものと見なされる。現在、言の葉ではこのような機能を用意していない。これは、今後いろいろな使用経験を経て、必要とあれば拡張したい。

また、LFGでは、ある素性値がある定数であることを要求するために
(↑INF)=c+
のように書くことが出来る。これは言の葉でも等式の右辺にXn以外の文字を書くことで指定できる。

しかし、LFGではある関係が成り立たないことを要求するために
¬[(↑SUBJ NUM)=SG.and.(↑SUBJ PERS)=3]
等のように書くことが出来る。このような否定の機能はいずれかならず必要になると考えられる。

6．2　言の葉の拡張

前節でも述べたように、言の葉では、今後拡張が必要と考えられる機能がいくつかある。実際に簡単な対話システムを構築して見て感じることは、F－リストを構築するための、各種の演算子が欲しいことである。言の葉の出力は、意味処理プログラムに引き渡されるが、その前に出来るだけ意味を素直に表わすF－リストを構成しておきたい。このことと、前節でも触れたような、条件を指定する制御機能を強化することによって、一種のプログラミング機能を、単一化規則に与える必要がある。

今後の考えられる拡張として、F標識を単一化規則によって与える方法を考える。例えば変数として%1，%2,……を使い、この変数に文字列を代入しておき、その値をF標識として使うことが考えられる。このような機能を使って、例えば図5．3の規則5)を図6．3のように書くと、

「亀の上位概念」に対するＦ－リストは図６．４のようになるであろう。

```
5) NP -> NP3 N
[  %1 = X3.SEM;
   X1.SEM.%1 = SUB;
   X1.SUB = X2;
];
```

図６．３　Ｆ標識を指定する規則

```
[SEM  [JouiGainen  [SUB;
                   ];
      ];
  SUB  [SEM kame;
      ];
];
```

図６．４「亀の上位概念」に対するＦリスト

このようなデータ構造の方が、関数関係を素直に表わす場合がある。

７．終わりに

自然言語をコンピュータに理解させる試みは、人工知能の中でも最も重要な研究分野の一つである。とくに日本語についても多くの研究がなされているが、必ずしも広く応用されている訳ではない。日本語の処理が、もっと容易にしかも確実に行なうことが出来れば、その応用範囲は非常に広いものとなるであろう。人間の理解過程そのものを研究する立場からも、また種々のアプリケーションプログラムに自然言語に近いインターフェースを与える立場からも、日本語をもっと手軽に扱う手段が望まれている。

これまでこうした企図を阻んでいたのは、日本語の文字種類が多かったこと、分かち書きをしないため、形態素解析が困難であること、そして日本語の文法が必ずしもコンピュータ処理に適する形で整備されていないこと等が考えられる。とくに文法を整備する事は、今後の日本語処理の発展を考える上で極めて重要な課題であるが、一朝一夕に解決する問題ではない。そのためにもいろいろな文法を仮定してコンピュータによる解析の実験を繰り返す事が必要と考えられる。

これまで文法の理論は、英語やヨーロッパの言語を中心に発達してきたものであり、その中から文脈自由文法CFGなども現れたものである。そのため、CFGよりは格文法の方が日本語の解析には適していると言う人もいる。しかしCFGについてはこれまでに膨大な量の研究がなされており、とくにその構文解析の方法論はほぼ完成されたものなので、可能ならばこれらの手法を使うべきであろう。さらに最近注目されつつある単一化文法はCFGの極めて自然な拡張となっており、これによって日本語の包括的な文法理論が構築される可能性もある。

本論文では、単一化文法によって記述した文法と辞書をデータとして読み込み、構文解析と単一化の処理を並行して行なうシステムを構築するためのオブジェクトのクラス群「言の葉」を提案した。言の葉では、入力文に含まれる複数の候補単語を全て三角行列に登録し、それに対してCKY法を適用する方法を採用している。これによって形態素解析のための各種の発見法の適用を

省略している。

この方法は著者が単独で発見したものであるが、ほぼ同じ方法が文献(3)にある。ただし文献(3)ではこの他に接続関係のチェックを行なっているのに対して本論文ではそのようなチェックを全く行なわない点が、やや異なる点である。

また、ボトムアップ並列なアルゴリズムであるCKY法との組み合わせを考慮して、句構造規則に付加する単一化規則は、下位のF構造を変化させないものとしている。

また本論文では、極めて小さい文法と辞書について例を示し、言の葉の出力から意味を取り出すことが比較的容易であることをも示した。

言の葉では未知語の処理を未だ組み込んでいない。従って、辞書に記載されていない単語については、何らかの発見法を適用して切れ目や品詞を推定する機構を今後組み込む必要がある。また、6．でも述べた様に、単一化規則を記述する方法についても拡張して行く必要がある。

言の葉は何か特定の文法を仮定し、それに基づいて文を解析するものではない。そうではなく、単一化文法による一定の書き方に従う限り、いろいろな文法を記述することが出来る点に大きな特徴がある。言い換えれば本論文では、いろいろな文法を記述するための、単一化文法による記述法とその処理系を提案しているのである。これによって、今後各種の文法を記述することが比較的容易に行なえ、それに基づいて各種の対話システムなどを実験的に構築することができる。そうした実験を通して言の葉を改良するとともに、いろいろな意味処理システムを構築するためにある程度共通に使える部分をオブジェクトのクラス群として見いだして行きたい。

引用文献

(1) 野村浩郷：“自然言語処理の基礎技術”，電子情報通信学会編，(1988)
(2) 野口正一監修，牧野武則著：“自然言語処理”，オーム社（1991）
(3) 田中穂積：“自然言語解析の基礎”，産業図書（1989）
(4) 田中穂積，辻井順一共編：“自然言語理解”，オーム社（1988）
(5) Bresnan J.,ed.：“The Mental Representation of Grammatical Relations”. The MIT Press (1982)
(6) チョムスキー著、安井　稔　訳：“文法理論の諸相”，研究社（1970）
(7) 黒川利明監修，東条　敏著：“自然言語処理入門”，近代科学社（1988）
(8) 渕　一博監修，古川康一，溝口文雄共編：“自然言語の基礎理論”，共立出版（1986）
(9) シャンク／リーズベック編，石崎　俊　監訳：“自然言語理解入門”，総研出版（1986）
(10) 郡司隆男著：“自然言語の文法理論”，産業図書（1987）