数理解析研究所講究録
第396巻 1980年 319-354

自然言語処理のためのソフトウェア構造

京大　工学部　辻井潤一
中村順一

1　自然言語処理の問題点

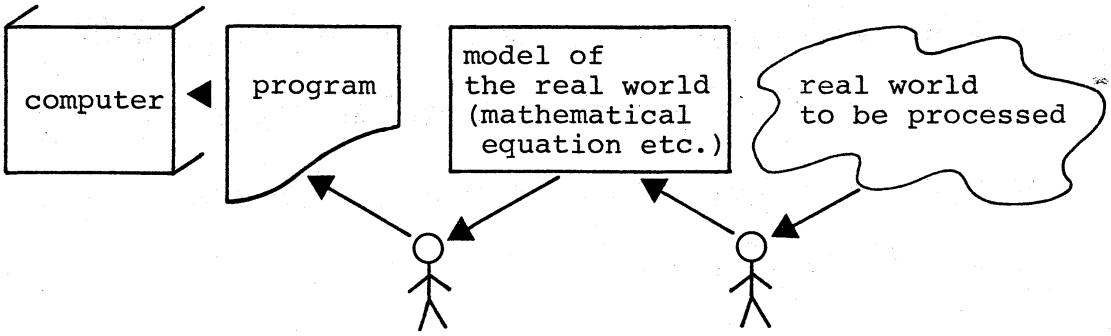
機械翻訳・自然言語によるデータ・アクセス等，自然言語（日本語，英語 etc.）を計算機で処理する試みが近年盛んになってきている。しかしながら，日本語・英語にあらわれる表現のかなりの部分を処理できるようなプログラムを系統的に構成してゆこうという試みは，これまでに多くのグループが試みているにもかかわらず，汎用的で強力な自然言語処理プログラムは現在まだ開発されていない。そこには，適切な自然言語のモデルの作成という言語そのものの研究とともに，ソフトウェア学的な立場から考察しなければならない多くの問題がある。

一般的にいって，これまでのプログラム作成手法は，まず

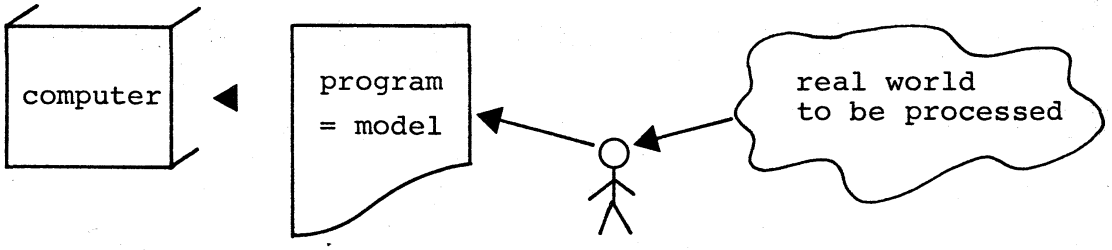
明確に処理対象を人間が把握し，これを計算機処理に向いた処理手順に人間が変換してから，プログラム化することを前提としていた。これに対して，structured programming や abstract data type にみられる stepwise refinement の手法は，対象問題の把握と，それの処理手順（プログラム）への変換との間にあった gap をできるだけ少なくすることによって，人間の問題把握の進み方を，そのままプログラムに反映してゆき，処理対象の把握と，プログラム作成の過程とを，同時進行的に行なってゆこうとするものであった。

しかしながら，一方では，この stepwise refinement の手法がうまく適用できない，‘複雑な’分野がある。医療診断，有機化合物の構造推定システムに代表される，知識工学が対象にしようとしている分野のプログラミングである。ここでは，処理対象の完全な把握そのものが，別の学問分野をなしており，したがって，システム作成の時点では，とても完全な処理対象の把握が不可能なような分野である。一度作成されたシステムは，その後も修正，改善されることが前提となっている。一度完成したシステムを保守するという従来の考え方よりもむしろ，未完のモデルやシステムを運用しながら，常に debug してゆき，この過程でモデルやシステムを完成に

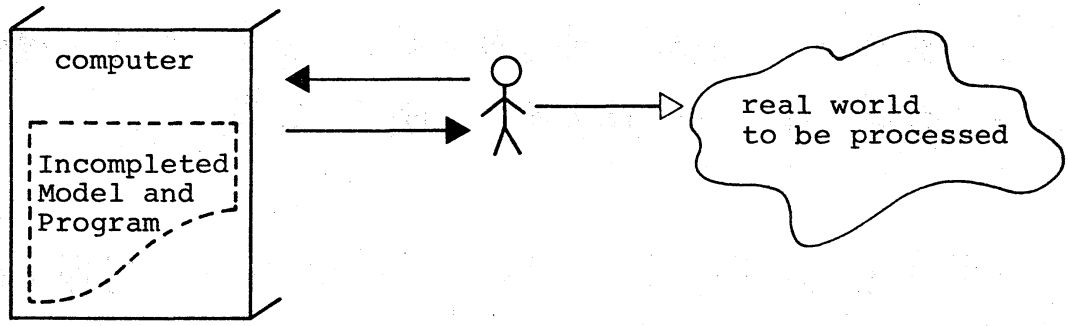
近づけてゆくと考える方が自然である。このような考え方を，知識工学的な考え方と呼ぶことにする。従来のプログラミング手法，stepwise refinement による手法，知識工学的手法の3者を模式的に図1に示す。



(1) Conventional Method of Programing



(2) Programing using new methodology
such as Stepwise Refinement etc.



(3) Programing Method in Knowledge Engineering

Fig. 1 Typical Frameworks for Programing

自然言語処理システムもまた，処理対象のモデルが，最後まで把握できないという意味で，知識工学的なアプローチが必要になる典型的なシステムである。すなわち，自然言語処理に必要となる知識 ― 自然言語の文法モデル ― は，それ自身言語学の研究課題であり，永久に解の得られる問題ではない。近似的モデルをまず作り，実験を繰り返すことによってこれを徐々に refine してゆくという evolutional なアプローチをとらざるを得ない。

自然言語処理を行なうためには，一般に次のような‘知識’を，データなりプログラムの形式でシステムに与えておく必要がある。

1。文法についての知識 ― 一般にどのような単語に関しても成立する文法規則
2。辞書 ― idiom・用例等の個々の単語に関する規則
3。意味，あるいは現実世界に関する知識

これら 1。～ 3。は，いずれもシステムにより管理される‘データ’であり，また入力文を処理する法則を記述している‘プログラム’でもある。以下では，この種の‘知識’が，

これまでの自然言語処理システムにおいて，どのようにencodeされてきたか[1]，またそれぞれの枠組はどのような問題点があったかを指摘し，現在我々が自然言語処理用のプログラミング言語として考えている枠組について触れることにする。

2　文法記述の基本的な枠組

2－1　プログラム

自然言語処理の研究は，歴史的には機械翻訳システムの研究を契機として開始された。初期のシステムは，自然言語の規則性をプログラムで把えようとするもので，機械翻訳システムSYST.RANがその代表である[2]。SYSTRANは，当時は言語処理用の特別なマクロ命令を持った，機械翻訳用のプログラミング言語といわれたが，現時点のソフトウェアから見ると，汎用のプログラミング言語と大差はない。

文法記述用の機能をあまり持たない，汎用のプログラミング言語で直接自然言語処理を行なうと，どのようになるかをみてみるために，SYSTRANでの処理の一部を以下に示

す。処理対象は，辞書引きがすでに終っている英文である。

[文の解析]

- 左から右へ scan し，多品詞語を見つける。
- 左右の単語をみて品詞を決定する。
- 品詞決定ができないときは，優先度情報によって決定する。
- イディオムを認定する。

[pass 0]

- relative pronoun, subordinate conjunction, punctuation sign等の文区切りを認定する。
- 従属節の型を決める。
- 関係代名詞より右の，最初の活用動詞を検出し，その右にコンマがあるかどうかを調べる。
- なければ，関係代名詞の左を見て，活用動詞を捜す。
- そこで関係代名詞から才2の動詞の手前までを1つの句として区切る。

[pass 1]

・文を右から左へscanし，明確な構文グループをまとめる（10種類）。
⋮

以下同様に，左から右へ文を何度もscanしながら処理が進められる。SYSTRANでは，単語の辞書に貯えられた各種の膨大な言語データの集積を個別のサブルーティンが参照してゆくことによって処理してゆくが，文法規則や辞書記述の使い方が非常にad-hocであることなど，言語情報とプログラムとが混然一体となっており，理想的なソフトウェアからはほど遠い。SYSTRANの欠点を整理すると，次のようになる。

(1) サブルーティンの集合体でシステムを構成しているため，ある程度以上の規模の文法になると，システムを保守できなくなる。この原因としては，

(1.1) サブルーティン間の，データを介した相互干渉

(1.2) 順序制御が基本であるため，新たなサブルーティンを全体の処理過程のどこで起動すべきかが決定できなくなってくる。ある場合には，入力文に応じて処理ステップの順序を入れ替えなければな

らないこともあり，順序制御によるシステムの致命的な欠陥となる。

(1.2)の欠点は，rule based system への移行によってある程度解消できる。しかし，(1.1)は，自然言語処理が単なる Production System の応用では済まないことを示している。

(2) 自然言語文の曖昧さに対する配慮がない。多くの自然言語文は，人間にとってさえ曖昧であり，複数個の解析結果を出す能力は，自然言語処理システムには不可欠である。また，大域的にみれば(文全体をみれば)，一意に解釈が決定できる文であっても，ある部分を局所的に解析している時点では，二様に解釈できることも多い。このことから，自然言語処理においては，処理過程での non-determinism が不可欠になるが，SYSTRAN では，これに対する配慮がないため，初期段階での処理の誤りが致命的となる。
⟶ backtracking, parallel, wait-and-see[3]

(3) 辞書記述を重視して，辞書中に文法的な規則性(多品詞語，多義語の disambiguation)を記述しているにもかかわらず，この記述を処理に反映するには，それを解釈する

ためのサブルーティンを用意し，処理過程中に適切に埋め込まなければならない。⟶一般規則と個別規則の取扱い

(4) 言語モデル・文法体系があまりにも ad-hoc である。

(4)の問題は，自然言語処理のソフトウェア体系には直接関係しないので，以下では(1)～(3)の各問題がそれ以降の研究でどのように把えられ，解決されてきたかを，代表的なシステムを例にして考えることにする。

2-2 ルール・システム

処理過程の順序依存性をなくす，最も直接的な方法は，プログラムによる表現の対極である，rule based system に移行することである。rule based system の典型は，形式言語やプログラミング言語の文法記述によく使われるCFG（Context Free Grammar）を，自然言語の文法記述にも適用することである。実際，CFGは，Kuno-Oettinger[4] の predictive analyzer をはじめ多くの自然言語処理システムに使われている。CFGの解析アルゴリズムについては，こ

れまでに多くの解説があるのでその詳細はここでは省略する。CFGは，SYSTRANの持っていた処理の順序依存という欠点を解消し，またアルゴリズムを工夫することによって，文法記述の時点では意識しなくても，複数個の解析結果が同時に得られる等，文法の保守容易性を向上している。しかしながら，CFGには次のような欠点があった。

(1) 統語的な規則性は，CFGの書き換え規則で自然に表現できる反面，それ以外の‘意味的規則性’や‘形態素的規則性’がうまく把えられない。同一の書き換え規則集合内で無理にこれを把えようとすると，無意味な非終端記号の増大を招き，文法の保守容易性が極端に悪くなる。

(2) これを避けるために，処理ステップを図2のように区別し，統語処理において参照すべき自然言語の規則だけを書き換え規則によって表記することが考えられるが，この方式では最初の処理ステップで行なわれる形態素処理や，その次の段階の統語処理において使うことのできる規則が制約条件としては弱すぎるために，統語処理が終了した段階ではいくつもの異なった解析木が組合せ的に生じることになり，事実上処理が不可能になる。

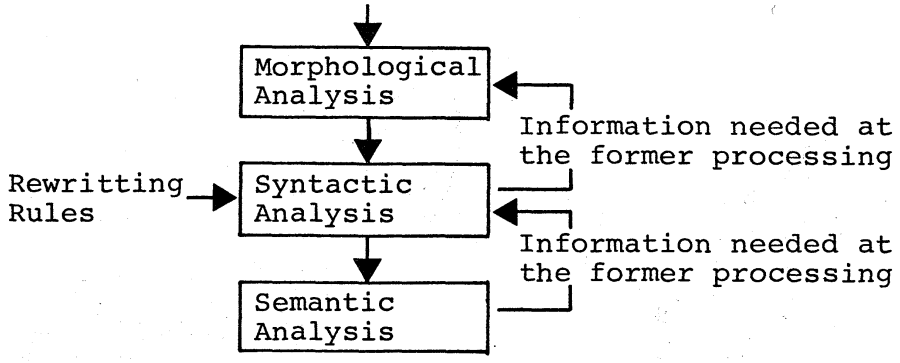


Fig. 2 CFG and the Processing Step

(3) 自然言語の統語的な規則性は，オー次近似としては品詞という抽象化したレベルで把えられるが，実際にはかなり各単語に依存した処理になる（例えば，Hornby の辞書においては，英語の動詞を，そのとり得る統語構造にもとづいて，25 のカテゴリに細分している）。したがって，統語的な規則性だけを考えている場合であっても，その中には各々独立に作用する規則（例えば，Hornby の動詞カテゴリにもとづく構文規則と，主語・述語の性数一致など）があり，これを把えるために非終端記号の組合せ的な増大をひきおこす。

(4) 自然言語の統語的規則性の中には，英語の wh-movement

規則に代表される変形規則があるが，これがCFGでは把えきれない（図3参照）。(5)

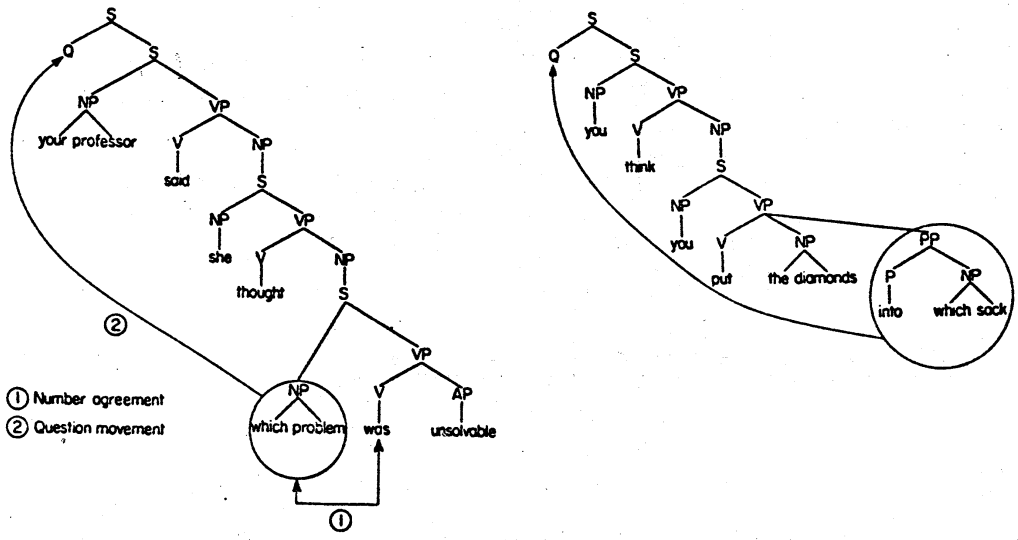


図3　変形規則 — movement 規則の例

結局，CFGは，rule based system であるために，文法規則の追加・変更が比較的容易にできるように見えるが，実際には書き換え規則によって表現できる言語の規則性が，自然言語全体の規則性の比較的一部であること，したがって，それ以外の規則性（意味的・形態素的規則性，変形規則，辞書規則 etc ）を書き換え規則の形式に無理に表現しようとすると，文法の保守容易性が極めて悪くなるという欠点を持っている。

2－3　TNあるいはPDA

CFGに対応して，そのCFGによって規定される文集合を認識するPush Down Automaton（PDA）を考えることができる。したがって，文法記述をCFGで与える代りに，PDAで表現することが考えられる[6]。この枠組も2－2のCFGとほぼ同じ欠点を持つと考えて良かろう。

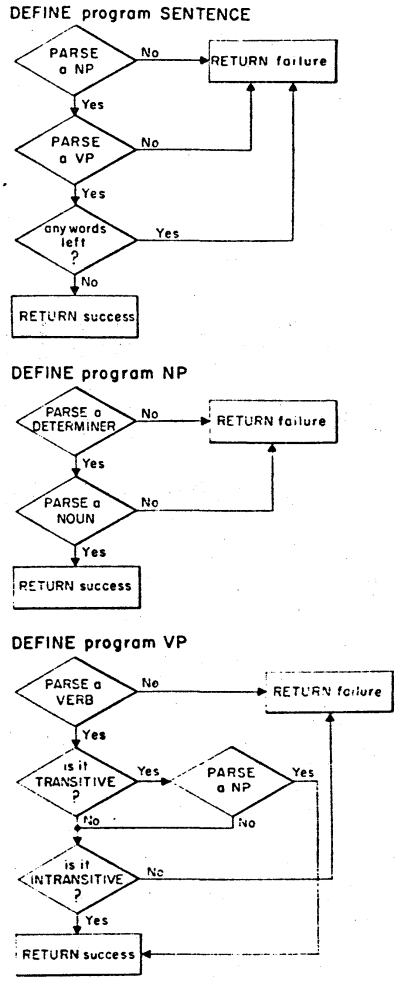
3　枠組の拡張

2節で述べた3つの基本的枠組は，それぞれに文法を系統的に作成してゆくには欠点があった。本節では，これらの欠点を補うために，それぞれの枠組でどのような試みがなされてきたかをふりかえってみることにする。

3－1　汎用プログラミング言語の拡張

自然言語処理に必要となる機能を，SYSTRANよりもさらに一層プログラミング言語の言語仕様の中にとり込み，プログラムによる文法の記述を容易にしてゆこうとする試み

がある。Winograd が，彼のＳＨＲＬＵシステムを開発するために使ったＰＲＯＧＲＡＭＭＡＲ，[7] Saarbrucken大学が機械翻訳システムのために開発した，ＰＡＳＣＡＬ－base のプログラミング言語ＣＯＭＳＫＥＥなどが，このカテゴリーに入る。ＰＲＯＧＲＡＭＭＡＲによる簡単な文法記述の例を図4に示す。ＰＲＯＧＲＡＭＭＡＲでは，構造解析を行な




```
2.1  (PDEFINE SENTENCE
2.2   (((PARSE NP) NIL FAIL)
2.3    ((PARSE VP) FAIL FAIL RETURN)))
2.4  (PDEFINE NP
2.5   (((PARSE DETERMINER) NIL FAIL)
2.6    ((PARSE NOUN) RETURN FAIL)))
2.7  (PDEFINE VP
2.8   (((PARSE VERB) NIL FAIL)
2.9    ((ISQ H TRANSITIVE) NIL INTRANS)
2.10   ((PARSE NP) RETURN NIL)
2.11  INTRANS
2.12   ((ISQ H INTRANSITIVE) RETURN FAIL)))
```

規則 1.6 から 1.9 は次の形となる：

```
2.13 (DEFPROP GIRAFFE (NOUN) WORD)
2.14 (DEFPROP DREAM (VERB INTRANSITIVE) WORD)
```

図4　PROGRAMMARによる文法記述

うために呼び出した下位ルーティンがその仕事に失敗した場合には，自動的に backtracking を行なう機能を持っており，システムとして non-deterministic な記述を可能にしている。これ以外にも，解析木を生長させたり，刈りとったりする操作をシステム関数として用意するなど，直接プログラムで文法記述を行なっても，文法自体の明晰さをおとさない工夫をしている。CFGと比べると，プログラミング言語という汎用的な枠組を用いているために，統語的な規則性以外のもの（Winograd のシステムにおいては主として意味的な規則）も，必要に応じて処理中に自由に参照できる。また，例からわかるように，プログラムの構成自体に文法モデルを直接反映することができるため，プログラムの可読性ははるかに向上している。

しかしながら，Winograd 自身が指摘しているように，PROGRAMMAR での文法記述においても，依然として可読性は十分でなく，自由に文法を変更したり，新たに修正を加えることは，ほとんどできない．すなわち，処理の順序依存性，procedure 間のデータ構造（解析木）を介した interface をすべて文法記述者がプログラム中に指定しなければならない，といった通常のプログラミング言語の持っている欠

点が依然として残っている。

この欠点を補うためには，‘より言語処理に向いた機能をプログラミング言語の仕様にとり込んでゆく’か，‘より rule based な system に移行する’か，が考えられるが，我々としては rule based system への移行が，不可決ではないかと考えている。

3－2　CFGの拡張

CFGによるシステムの欠点は，自然言語の持つ規則性の一部（統語的規則性）を表現するのに適した記述形式によって，自然言語のすべての規則性を表現しようとすることにあった。この欠点を補うために開発されたシステムがいくつかある。Augmented CFGと呼ぶべきカテゴリである。このカテゴリに属するシステムとしては，LINGOL(MIT)[8][9]，拡張LINGOL(ETL)[10]，MEILING（東大），EASY（京大），APS（IBM）[11]，DIAGRAM(SRI)[12]等がある。これらのシステムでは，統語的な規則をCFGで表現し，それ以外の規則性は，各々の書き換え規則につけられたprocedureによって表現する。例えば，拡張LI

ＮＧＯＬでの文法規則は次の５字組で表記される。

(<left> <right> (<advice> <cog>) <sem>)

ここで<left>, <right>の対が，書き換え規則

<left> ⟶ <right>

を表現している（ただし，ＬＩＮＧＯＬでの書き換え規則はChomsky Normal Formに変換されており，<right>には高々２つの非終端記号しか書けない）。<advice>,<cog>, <sem>の各部分が，書き換え規則をaugmentしている部分であり，ここに埋め込むprocedureによって統語的規則性以外の規則性を処理する。

<advice>部に指定されたprocedureは，書き換え規則を実行する際に評価され，意味的・形態素的にみて，その規則を適用することが正当であるかどうかをチェックする。<cog>部は，入力文のある部分に対する解釈が複数個生じた場合に評価され，最も妥当だと思われる解釈を選択するために使用される。また，<sem>部はＣＦＧによって作られた解析木を，意味構造へ写像するためのプログラムが指定されている。一種のsyntax directed translationを行なうための機構で，入力文に対する全体的な解析木が構成された後に評価される。

CFGをprocedureでaugmentするもう一つの例として，SRIのDIAGRAMをみてみよう。DIAGRAMでは，LINGOLの<advice>，<cog>，<sem>がそうであったように，書き換え規則がシステムによって適用されてゆく過程で，それぞれ異なったタイミングで評価される2つのprocedure記述（constructor，translatorと呼ぶ）で書き換え規則をaugmentしている。constructorは，LINGOLの<advice>と同様に，規則が適用されると同時に評価され，規則適用の可否をチェックする。一方，translatorは，<sem>に対応し，入力文の解析が終った段階で起動される。DIAGRAMでの規則の例を図5，図6に示す。

書き換え規則の不備をprocedureで補うこれら一連のシステムは，基本的にはrule based systemになっており，システムのevolutionalな開発という点ではすぐれた形式になって

```
Rule SB1          SDEC = NP BE PRED :

    Constructor: (1) If NP and BE do not agree in syntactic
                     number, then REJECT the analysis.
                 (2) If either BE contains "not" or NP
                     directly dominates a determiner phrase
                     containing "no," then SDEC is Negative:
                     else it is Affirmative.

    Translator:  (1) If the syntactic number of NP is null
                     (undefined), then set it equal to the
                     syntactic number of BE.
                 (2) Set NP as Subject of SDEC.
```

図5　DIAGRAMでの規則 (1)

```
(NP1      NP = (D=[A / DDET / DETQ]) NOMHD (NCOMP) ;
     CONSTRUCTOR (PROGN (COND
                          ((@ D)
                            [COND
                              ((MASS? D)
                                (OR (MASS? NOMHD)
                                    (F.REJECT (QUOTE F.MASS]
                            [COND
                              ((MASS? NOMHD)
                                (OR (NOT (@ A))
                                    (F.REJECT (QUOTE F.MASS]
                            [COND
                              ((@ NCOMP)
                                (@SET NBR (@INTERSECT NBR D NOMHD NCOMP)))
                              (T (@SET NBR (@INTERSECT NBR D NOMHD]
                            (@FROM D DEF NOT))
                          ((AND (SG? NOMHD)
                                (NOT (MASS? NOMHD)))
                            (@FACTOR (QUOTE F.NODET)
                                     UNLIKELY))
                          ((@ NCOMP)
                            (@SET NBR (@INTERSECT NBR NCOMP NOMHD)))
                          (T (@FROM NOMHD NBR)))
                        [AND (@ THANCOMP NCOMP)
                             (OR (@ THANCOMP NOMHD)
                                 (F.REJECT (QUOTE F.THANC]
                        (@FROM NOMHD TYPE)))
```

図6　DIAGRAMでの規則(2)

いる。しかしながら，共通の欠点として

(1) 統語的な規則性以外はすべて user 定義の procedure に埋め込まれることになり，文法記述の多くの部分が procedure という black box の中にかくれてしまう。

(2) 意味的処理・変形処理といった自然言語解析のかなり大きな部分を占める操作が，ＬＩＮＧＯＬの<sem>，ＤＩＡＧＲＡＭの translator のように，全体の解析木が作られた後に起動される procedure によって実行されることに

なる。すなわち，統語処理の終了後に，意味処理・変形処理を行なわなければならないというCFGの従来の欠点がそのまま残っている。

(3) 一般的な文法規則と単語ごとの規則とに区別があり，単語ごとの規則を強いて反映しようとすると，非終端記号を無意味に増加させるか，埋め込まれたprocedure中で，単語の辞書記述を参照するしか方法がない。

このように，Augmented CFGは，augmentする以前の従来のCFGシステムに見られた欠点を，何らかの形でほぼそのままひきついでいる。ただ，procedureを各書き換え規則に埋め込むことによって，文法記述者が工夫すればなんとかこれを補えるようになっている。しかしながら，文法規則や辞書記述を計算機システムが管理し，evolutionalな形式で自然言語処理システムを開発してゆこうとした場合に，記述の多くの部分がprocedureというblack box中にimplicitに埋もれてしまうことは，致命的である。この欠陥を補うために，ETLの田中らは，LINGOLとは独立に，意味的規則性等の，書き換え規則では把握できないものを，より記述的に表現する手段として，Bobrow・WinogradのKRL[13]と

類似の notation を持った SRL[14] を開発しているが，LINGOL で表現された文法と SRL での記述とが，うまく統合されるまでには致らず，(2)・(3)の欠点はそのまま残っているといえよう。

3－3 TN の拡張

TN の拡張（Augmented TN，ATN）は，CFG の拡張とほぼ並列的に行なわれている[ATN（BBN）[15][16][17]，PLATON（京大）[18]]．すなわち，PDA の state-transition の際の条件チェックに，従来の CAT，WORD 等を使ったチェックだけでなく，user define の関数チェックを使えるようにすること（LINGOL の <advice>，DIAGRAM の <constructor> に対応），また解析木の構造を組みあげる際に，それまでにセットしておいた変数（レジスタ）を自由に組み合せることを許すこと（<sem>，<translator> に対応）等である。したがって，ATN においては，wh-movement 等の変形規則が，HOLD 操作によって枠組内で自然に，明示的に表現できる以外は，Augmented CFG の持つ欠点，すなわち，自然言語の持つ規則性の多くの部分が user define の procedure 中に implicit な形で埋め込まれ

る欠点は，ほぼそのまま残る。

3－4　Tree Grammar による拡張

CFGの問題点は，規則によって組み上げられた部分木構造についての情報が，すべてその部分木のrootにつけられたlabel（非終端記号）に集約されてしまう点である。したがって，もし書き換え規則を，終端記号・非終端記号の列に対して適用するのではなく，それまでに組み上げられた部分木構造の列に対して適用することにすれば，より豊かな情報にもとづいて規則を適用することができる．また，解析用文法だけでなく，機械翻訳システム等に適用される文法は，生成用文法・変換用文法もあり，これらの記述に際しては木構造の変換操作が不可欠となる。変形規則も木構造に対する操作として容易に定式化できる。このような発想で作られたシステムにCETA[19]，ROBRA[20]（グルーブル大学），Q－SYSTEM[21]（モントリオール大学），PLATON（京大）がある。図7にROBRAでの記述例を示す。この木構造変換という枠組で文法記述をとらえることは，procedureの埋め込みによってCFG，TNをaugmentする手法に比べると，言語の持つ規則性がよりexplictに文法記述上にあらわ

```
R1 : (*3., niv ≤ 3)    1.AB (?2.AB &B (*) 3.&ND (A, *, B($L)))/
                       2? = BC & 3. ∈ {CD, XX}

     =>    3.(C, $L,  4.(A), 1.) / * ← B., 1. ← A ; C, 4., A ← * /
                       4. := 3. ; 1. := 3.
```

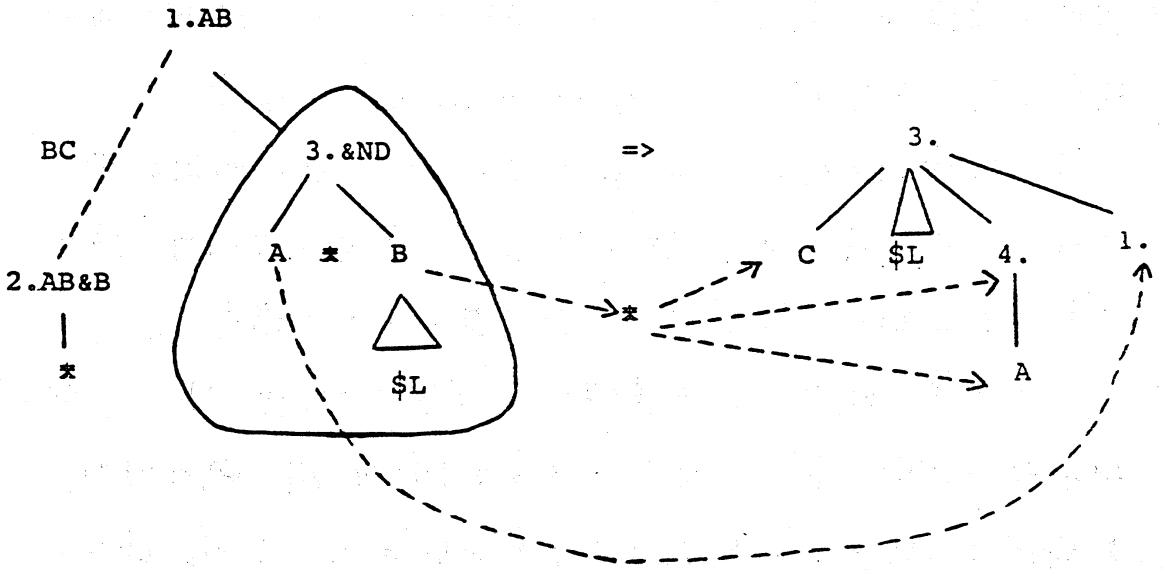


図7　プログラミング言語ROBRAでの記述例

れるという利点がある。

また，部分木構造の情報が，木構造のrootにつけられたlabelだけに集約されるという欠点は，Knuthがcfgの意味記述として提唱した属性文法のように，解析木の各節点にlabelだけでなく，属性集合を与えることによっても解消できる．ROBRAでは，この種の機能も使っている。

3－5　単語ごとの記述の重視

現在の言語学の最もよく使われている枠組に，Chomsky[22]らの変形文法理論があるが，この枠組においても一般的な規則（構造依存的な規則 ― structure dependent rule）は，ごく限られた規則を除いてほとんどなくなり，意味解釈規則と呼ばれる，各単語個別の規則によって，言語の規則性を把えようとする傾向になっている。木構造とその木構造につけられたカテゴリー記号だけを基礎にした規則で説明できる言語の規則性は非常に狭い。適用できる範囲が動詞一般の規則，他動詞だけの規則，他動詞の中でも目的語としてthat節をとる動詞だけの規則，そして最終的にはidiomのように，各単語固有の規則，というように，言語の規則には，その適用範囲に種々の階層性を考えることができる。

この階層性を，augmentationなしのCFGで表現しようとすると，動詞というカテゴリーをさらに細かく分類し，その分類に対する非終端記号を使ってCFG文法を作ることになろう。文法モデルを精密にしてゆこうとすればするほど，個別単語についての規則や少数の単語クラスについてのみ成立するような文法規則が増加してゆくことになるために，非

終端記号の爆発的な増大を招くことになる。文法モデルの精密化が，文法規則の exponential な増大をひきおこし，文法の保守容易性が極端に悪くなる。

一般規則から出発し，これを徐々に精密化してゆくというアプローチとは対照的に，個々の単語ごとに規則を記述しておき，これを使って文構造の解析を行なう立場がある。この立場を徹底的に追究したのが，Rieger の Word Expert Parser[23] である（図8）。ただし，この Rieger の Word Expert Parser では，一般的な文法規則が全くないこと，単語ごとに与えられた記述を適用してゆく一般的な機構をシステムが持っていないために，解析過程のすべての制御を各単語ごとの Expert が co-routine 的な制御機構を使って行なわなければならず，文法のわかり易さは極端にわるくなる。

したがって，一般規則中心の従来のシステムに，単語個別の規則を自然に書き込める工夫を入れ込むことで，従来のシステムを augment することが考えられる。この種の試みに，ATN と semantic network を結合した Laubsch のシステムがある[24]。図9に Laubsch のシステムでの記述例を示す。

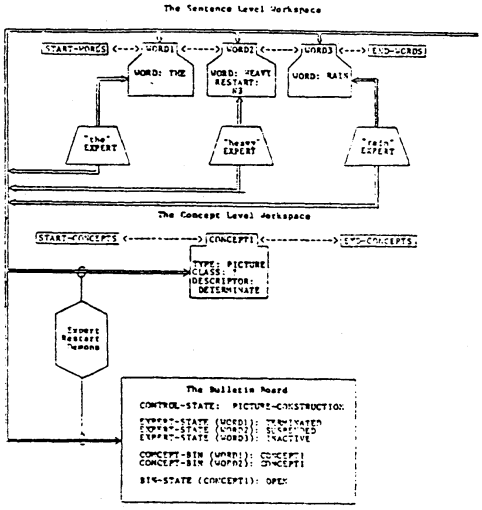


図8 Rieger の Word Expert Parser の概念図

```
(Concept consumption
   (Individuates Quant/attr/prod) ; an attribute
   (Generic-properties
      of/product: (class PRODUKT)
      consumed/quantity: (class QUANTITY) ...)
   (Trigger-atn
    (triggered-by: (Words verbrauch- hoch Hoehe ...))
     [Q/PROP/OBJ ; subATN for query of object's property
         ((($!OBJ1 (PROP: $?PROP))
          ($!OBJ2 (PROD/CODE: $?PROD))) ¦ filter
          (Ask PRODUKT make: P with: $=,OBJ1)
          ; make an exemplar of a prod/code product
action ¦ (Ask self set: of/product: $=P)
          ; retain it in my of/product slot
         (Ask P prop: $?PROP)
          ; ask it how much it consumes on that
          ; attribute ;]))
```

図9 ATN と semantic-network：
単語固有の規則性の記述

4 議論の整理

これまでの議論を整理すると，次のようになる。

(1) 自然言語処理システムは，処理対象のモデル自体が明確でなく，計算機もプログラムをつくる時点で，処理過程についてすべてのことが明らかになっているわけではない。したがって，システムは，最初に作ったモデルを，実験をくり返し，新たな言語現象が明らかになった時点で，容易に変更できることが必要である。

(2) (1)の性質を持つような分野のプログラミング手法としては，知識工学の分野で発展してきた rule based system の考え方が有効である。もちろん，これまでの知識工学の分野でよく使われてきた Production System の枠組がそのまま，自然言語処理に適用できるとは思わないが，すくなくとも，処理に必要な知識をできる限り記述的に，また明示的に表現できるような，枠組を開発する必要がある。

(3) このような観点から見ると，従来の文法記述の方法，すなわちプログラムによる表現，ＣＦＧ・ＴＮによる表現は，

いずれも欠点を持っていた。また，これらの枠組をaugmentした手法も，元の枠組の欠点がいくらか矯正されているが充分ではない。

M. Kay（Xerox）は，最近の論文で'Functional Grammar'の見方にもとづいた新しい文法記述のnotationを提案している。(25) このnotation自体は，現在まだideaの段階で，計算機でこれを解釈し，実行するためには，解釈しておかなければならない問題が多くあるが，同時に，これまでの枠組を次のような点でaugmentしている。

(1) 自然言語処理の中に，data type的な考え方をうまく取り入れている。

(2) 木構造もdata type中の1つのfield値として表現しておくことにより，ROBRA，PLATON，Q-SYSTEM等の持っていた木構造列のパターン・マッチングと，属性値のマッチングとを，より高い立場から，同一の枠組で把えることができる。

(3) data type中のfieldとして，木構造的な値をとるもの

が複数個あることを許せるので，1つの記述の中に，複数個の木構造が共存することがあっても良い。このことは，ROBRA・EUROTRAの主張するmulti-level treeの一般化になっており，統語構造を示す木構造，意味構造を共存させることができる（図10）。このことは，これまでの統語構造を表現する全体の解析木をまずつくり，それから意味への写像を行なうという，stepwiseな処理をしなくても良いことを示す。同一の枠組で，統語情報・意味情報を区別なく表現でき，処理できるための利点である。

(4) LINGOL，DIAGRAM等のrule based systemにおいて，procedureでaugmentしていた多くの部分を，規則中に明示的に記述できる。したがって，この記述の部分をも，（High Level Language Oriented Editorを含む）シ

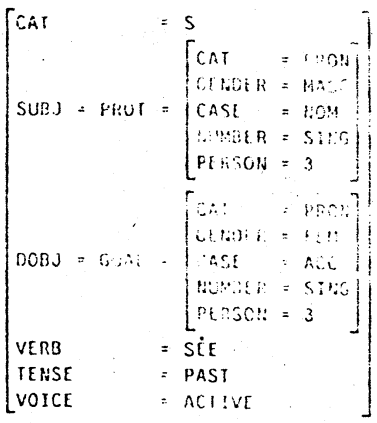


図10 統語構造木と意味構造の共存

ステムの管理下におくことができ，保守性を極めて高くすることができる。また，*procedure*を介在させないことによってPROLOGのような記述的な言語の持っていた，記述の双方向的な利用も可能になる。

(5) 単語固有の規則，辞書規則が一般規則と全く区別なく表現できる。このことは，一般的な統語規則と辞書規則との*interaction*を，文法記述者がそれほど意識しなくても良いことを意味している（図11）。

```
[CAT = VERB
LEX = persuade
↑VERB = [BENEF = NONE]
<↑VERB SCOMP SUBJ> = <↑VERB GOAL> = ANY]
```

図11　単語の個別規則の表現例

このように，この枠組はいくつかの点で，従来の枠組を拡張する可能性をもっている反面，文法記述者の書いた文法を解釈・実行するシステム側にそれだけ多くの負担をかけることを意味している。従来のCFGでは，*label*の一致だけを手掛りに処理を進めてゆけたのに対して，この枠組では規則を適用するに際しては，常に複合的な記述集合間のマッチング（この過程は，木構造のマッチングをも含んでいる）を行なうことになり，効率上問題となる。

また，現時点では，field値としては記号の使用しか許しておらず，そこでのマッチングも記号間のdirect matchingに限られている。ROBRAの場合でも，マッチングの際に簡単なboolean操作だけでも不充分で，user defineのprocedureまで許す必要が生じてくるかもしれない。全体の記述的枠組を破ることなく，user defined procedureを導入できるかどうか。これらの問題を含めて，我々のグループでは，このKayの提案をプログラミング言語として定式化する作業を現在行なっている。‘利用者の修正したデータセットを指定する’をこの枠組でhand simulateしたものを，図12に示す。

5 おわりに

自然言語処理の抱える問題は，大きくわけて言語現象の適切なモデル化の問題と，そのモデルを自然に表現でき，以後のシステム開発・運用を通じて，これを簡単に精密化してゆけるプログラミング環境を設定するという，ソフトウェア構成の問題とがある。本稿では，言語現象の詳細には触れずに，ソフトウェア的な面に限って議論してきたため，若干抽象化しすぎた面がある。また，全体的な展望というよりも，現在

の我々のグループの意見が，かなり強く反映していることも，つけ加えておく。

```
KEQ52800I  AMD1388.GENGO1.TEXT(N05)                    17:05:24     05/30/80
00010 ------------------------------------------------------------------
00020     *  Sample description
00030              Input sentence is 'Riyosha no shusei si ta dataset
00040                                    wo shitei suru'.
00050 ------------------------------------------------------------------
00060      *     Description for sentence
00070
00080     | KEY = : CAT = S
00090     |
00100     |    | CAT = S
00110     |    | PATTERN = | PRE = (?1 VERB)
00120     |    |           | POST = ( <*> )
00130     |    | VERB = | CAT = VERB
00140     |    |        | INF = shusi
00150     |
00160     |    | CAT = NP
00170     |    | PATTERN = | PRE = (?1 VERB-1 HEAD ?2 VERB-2)
00180     |    |           | POST = (?1 <*> ?2 VERB-2)
00190     |    | VEBR-1 = | CAT = VERB
00200     |    |          | INF = rentai
00210     |    | VEBR-2 = | CAT = VERB
00220     |    |          | INF = shusi
00230     |    | HEAD = : CAT = NP
00240     |    | MOD = | CAT = UMEKOMI1
00250     |    |       | VERB = <^MOD VERB-1>
00260     |    |       | HEAD = GAP = <^MOD HEAD>
00270     |    |
00280     |    |       | CAT = UMEKOMI2
00290     |    |       | VERB = <^MOD HEAD>
00300     |    |       | HEAD = <^MOD VERB-1>
00310     |    |       | GAP = NONE
00320
00330      * Description for NP
00340
00350     | CAT = NP
00360     | PATTERN = | PRE = (?1 HEAD ?2)
00370     |           | POST = (?1 <*> ?2)
00380     | <*> = HEAD = : CAT = NET
00390
00400 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
00410
00420      * Description for the input sentence
00430
00440     * riyosha
00450
00460     | CAT = NET
00470     | LEX = riyosha
00480
00490     * no
00500
00510     | CAT = KAKU
00520     | LEX = no
00530
00540     * shusei
00550
00560     | KEY = | CAT = SNOUN
00570     |        | LEX = shusei
00580     |
```

図12 (a) "利用者の修正したデータセットを指定する"の例

```
00590     CAT = VEBR
00600     PATTERN = | PRE = (?1 KEY SURU ?2)
00610               | POST = (?1 <*> ?2)
00620     SURU = | CAT = VERB-SURU
00630            | LEX = suru
00640            | INF = ANY
00650     LEX = shuseisuru
00660     INF = <* SURU INF>
00670
00680       <^VERB CAT> = UMEKOMI1
00690       <^VERB VOICE> =   ACTIVE
00700                         NONE
00710       <^VERB SLOT-1> =
00720             CAT = NP
00730             <^VERB PATTERN> =
00740                  | PRE = (?1 <*> KAKU ?2 KEY ?3)
00750                  | POST = (?1 ?2 KEY ?3)
00760             KAKU = | CAT = KAKU
00770                    |        | LEX = sa
00780                    |        | <^VERB GAKAKU> = <^VERB SLOT-1>
00790                    |        | LEX = no
00800                    |        | <^VERB NOKAKU> = <^VERB SLOT-1>
00810             <^VERB GAP>
00820             NONE
00830
00840       <^VERB SLOT-2> =
00850             CAT = NP
00860             <^VERB PATTERN> =
00870                  | PRE = (?1 <*> KAKU ?2 KEY ?3)
00880                  | POST = (?1 ?2 KEY ?3)
00890             KAKU = | CAT = KAKU
00900                    | LEX = wo
00910                    | <^VERB WOKAKU> = <^VERB SLOT-1>
00920             <^VERB GAP>
00930             NONE
00940
00950   * suru
00960
00970   KEY = | CAT = VERB-SURU
00980         | LEX = suru
00990         | INF = renyo
01000
01010       CAT = VERB
01020       PATTERN = | PRE = (?1 VMOD KEY ?2)
01030                 | POST = (?1 VMOD <*> ?2)
01040       VMOD = | CAT = VMOD
01050              | <*> = KEY
01060
01070       CAT = VERB
01080       PATTERN = | PRE = (?1 KAKU KEY ?2)
01090                 | POST = (?1 KAKU <*> ?2)
01100       KAKU = | CAT = KAKU
01110              | <*> = KEY
01120
01130       CAT = VERB-SURU
01140       PATTERN = | PRE = (?1 KEY ?2)
01150                 | POST = (?1 <*> ?2)
01160       <*> = KEY
01170
01180   * ta
01190
01200   | KEY = | CAT = JODO
```

図12 (b)

[1] Grishman, R.: A Survey of Syntactic Analysis Procedures for Natural Language, AJCL, Microfiche 47, 1976

[2] Slype, G. V., Pigott, I.: Description du Système de Traduction Automatique SYSTRAN de la Commission des Communautes Européennes, 1979

[3] Marcus, P. M.: A Theory of Syntactic Recognition for Natural Language, MIT Press, 1980

[4] Kuno, S.: The Predictive Analyzer and a Path Elimination Technique, CACM, 8, 1965, 687-698

[5] Bresnan, J.: A Realistic Transformational Grammar, in Linguistic Theory and Psychological Reality, (eds. Halle, Bresnan, Miller), MIT Press, 1978

[6] Thorne, J.P. et. al.: The Syntactic Analysis of English by Machine, in Machine Intelligence 3, Edinburgh University Press, 1968

[7] Winograd, T.: Understanding Natural Language, Academic Press, 1971

[8] Pratt, V. R.: A Linguistic Oriented Programming Language, IJCAI3, 1973, 272-381

[9] Pratt, V. R.: LINGOL-A Progress Report, IJCAI4,1975, 422-428

[10] Tanaka, H., Sato, T. and Motoyoshi, F.: Predictive Control Parser : Extended LINGOL, IJCAI6, 1979

[11] Heidorn, G. E.: Augmented Phrase Structure Grammar, in Schank and Nash-Webber (eds.): Theoretical Issues in Natural Language Processing, Cambridge, Mass. 1975, 1-5

[12] Robinson, J. J.: DIAGRAM: A Grammar for Dialogues, SRI Technical Notes, 1980

[13] Bobrow, D. G. and Winograd, T.: An Overview of KRL, a Knowledge Representation Language, Studies in Cognitive Science, Vol. 1, No. 1, 1977

[14] Tanaka, H., Sato, T. and Motoyoshi, F.: EXPLUS-A Semantic Parsing System for Japanese Sentences, 3rd USA-JAPAN Computer Conference, 1978, 236-240

[15] Woods, W. A.: Transition Network Grammars for Natural Language Analysis, CACM, Vol. 13, 1970

[16] Bates, M.: The Theory and Practice of Augmented Transition Network Grammars, in Bolc, L. (ed.): Natrual Language Communication with Computers, LNCS-63, 1978, 191-260

[17] Kaplan, R. M.: Augmented Transition Networks as Psychological Models of Sentence Comprehension, Artificial Intelligence, 3, 1972, 77-100

[18] Nagao, M. and Tsujii, J.: Analysis of Japanese Sentences Using Semantic and Syntactic Information, AJCL, Microfiche 41, 1976

[19] Chauche, P.: The ATEF and CETA systems, AJCL Microfiche 17, 1976

[20] Boitet, C. et. al.: Manipulation d'Arborescences et Parallélisme : Système ROBRA, in Proc. of COLING 78, 1978

[21] Colmerauer, A.: Les System-Q ou un Formalisme pour Analyser et Systhétiser des Pharses sur Ordinateur, Publ. Interne No. 43, Faculté des Sciences, Université de Montreal, 1970

[22] Chomsky, N.: Reflections on Language, Pantheon, 1975

[23] Rieger, C. et. al.: Word Expert Parsing, in Proc. of 6th IJCAI, 1979

[24] Laubsch, J. H.: Interfacing a Semantic Net with an Augmented Transition Network,in Proc. of 6th IJCAI, 1979

[25] Kay, M.: Functional Grammar, Xerox, PARC report