人工知能と対話技法 23－2
（1981. 11. 26）

モンテギュー文法に基づく英文和訳システムについて†

西田豊明，堂下修司 （京都大学 工学部 情報工学教室）

1. はじめに

本稿では，モンテギュー文法の枠組に基づいて英文和訳を行なうシステムについて，基本的な翻訳原理と設計方針を中心に報告する。モンテギュー文法は，自然言語表現を内包論理式に変換し，与えられたモデルの中でその内包論理式の解釈を与える，真理条件的，モデル理論的，可能世界の概念を用いた，意味論である[1]。本研究の目的は，このような意味論を中心とした数理的な枠組によって機械翻訳の過程を体系的に記述するとともに，それに基づいて深い処理を行なうことにより英語と日本語のように言語体系の大きく異なる言語間の翻訳に役立てようという実際的興味と，従来英語の小さな断片についてしか定義されていなかったモンテギュー文法の枠組を，機械翻訳という実際的処理を通して，より広い範囲について検証しようという理論的興味からなる。

はじめに，モンテギュー文法の枠組をどのように機械翻訳に適用するかについて原理を示す。これにより機械翻訳に関して，従来の手続き的なモデルではなくて，関数型のモデルが得られることを示す。次に，英文解析を行なうサブシステムと日本語生成のサブシステムについて述べ，最後に翻訳例を示す。

2. モンテギュー文法と機械翻訳

2.1 モンテギュー文法の枠組

図1にモンテギュー文法の枠組の概念図を示す。あいまい性を含みうる自然言語の表現は，統語規則によって，無あいまい言語とよばれる統語記述用の言語の表現の集合に対応づけられる。そしてその各々は，翻訳規則によって内包論理式に変換される。内包論理式は与えられたモデルのもとで指定された可能世界において解釈規則に基づいて意味解釈され，指示値として{真，偽}のいずれかを返す。モンテギュー文法の枠組では，文の意味（cf. 指示値）は，可能世界×文脈 → 真理値への関数として捉えられている。

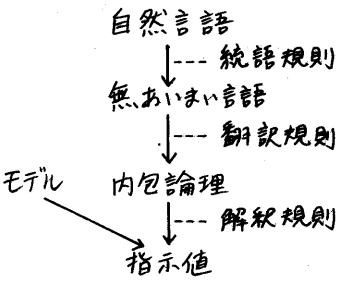


図1. モンテギュー文法の枠組

具体例を示そう。簡単な英文，

```
This command needs no operand.   ... (1)
```

を考える。基本表現と統語規則が図2のように与えられているとする。このとき(1)を統語的に解析すれば，(1)に対応する無あいまい言語表現の集合は，

```
{ F2(F1(this,command),F3(needs,F1(no,operand))) } ...(2)
```

であることがわかる。この場合，(1)はあいまいでないと考えられるので(2)は1個の要素しか含まない。

基本表現

NOUN = {command, operand, ...}
DET = {this, no, ... }
VT = {needs, ... }

統語規則

S1: $\alpha \in$ DET, $\beta \in$ NOUN
$\alpha\beta$ = F1(α,β) $\in$ NP

S2: $\alpha \in$ NP, $\beta \in$ VP
$\alpha\beta$ = F2(α,β) $\in$ S

S3: $\alpha \in$ VT, $\beta \in$ NP
$\alpha\beta$ = F3(α,β) $\in$ VP

図2. 基本表現と統語規則

†) 本研究の一部は文部省科学研究費による

次に (2) の要素に対して図3のように定義された翻訳規則を適用すると,

F2(F1(this,command),F3(needs,F1(no,operand)))
⇒ (this'(command'))
(λx[(no'(operand'))(λy[need'(x,y)])]) ...(3)
⇒ ∃x[∀y[x=y ↔ command'(y)] ∧
~∃z[operand'(z) ∧ need'(x,z)]] ... (4)

となる。論理式 (4) の意味は次のように理解することができる:

(4), すなわち文(1)が真であるための必要十分条件は,

与えられた状況において, ある x が存在して
任意の y に対して x=y となるための必要十分条件は y が command' であることであり
かつ, その x に関して
operand' でありかつ x need' え であるようなえが存在しない
... (5)

ことである。

(a) 基本表現に対して

NOUN ⇒ $Type_{<0,1>}$
command ⇒ command'
operand ⇒ operand'

DET ⇒ $Type_{<<0,<0,1>>,<0,1>>}$ |
this ⇒ this' ≜λp[λq[λy[∀x[p(x) ↔ x=y] ∧ q(y)]]]
no ⇒ no' ≜λp[λq[~∃x[p(x) ∧ q(x)]]]

VT ⇒ $Type_{<0,1,1>}$
needs ⇒ need'
ここで $Type_{<\alpha,\beta_1,\cdots,\beta_n>}$ は $\beta_1 x \cdots x \beta_n \to \alpha$ の写像(関数)のクラスを表わす.

(b) 統語操作に対して

Tr1: F1(α,β) ⇒ α'(β')
Tr2: F2(α,β) ⇒ α'(β')
Tr3: F3(α,β) ⇒ λx[β'(λy[α'(x,y)])]

図3. 翻訳規則

2.2 モンテギュー文法の機械翻訳への適用

内包論理の枠組は個々の自然言語に依存せず, 普遍的なものであると考えられる。内包論理式に含まれる定数に対して, 目的とする自然言語 (Target Language, TL)での解釈を与えると, その内包論理式に対応するTLの文を得ることができる。例えばTLとして日本語を選び, command' → コマンド, operand' → オペランド, needs' → 必要とする, … というような知識を与えると (5) に対応する日本語文を得ることかできる。しかし, これでは訳文としてぎこちなさすぎる。この原因は no や this に対する語彙分解にある。語彙分解が行なわれる以前の (3) を出発点とすれば, 図4のようにして, 日本語文を得ることができる。見方をかえてみると図4で行なっていることは, 通常われわれが英文和訳に関して持っている知識, 例えば, "no" を「… のような ~ はない」, 「一つの ~ も … ない」と訳すという知識を, 入表現によって厳密に記述したのにすぎない。

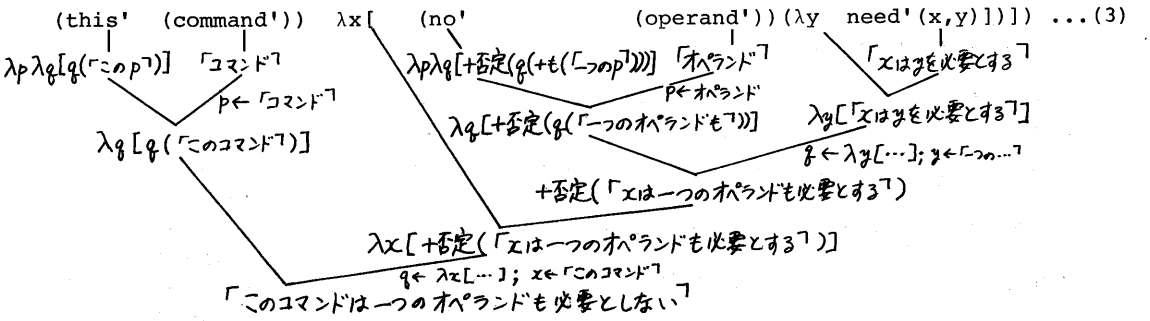


図4: 内包論理式 (3) からの日本語生成

2.3 モンテギュー文法に基づく翻訳の利点

モンテギュー文法の特徴は，準同型写像の原理，

$$\xi[\, f(\alpha_1, \ldots, \alpha_n)\,] = \xi[\, f\,](\, \xi[\alpha_1], \ldots, \xi[\alpha_n]) \qquad \ldots (6)$$

に要約される。ここでξはモンテギュー文法の各段階で用いられる写像である。機械翻訳に準同型性の原理を用いると，システムのモジュラリティ，理解度，簡単化の点で有利である。英日機械翻訳においては次のような場合に利点がある。

(例) prenominal negation

モンテギュー文法の場合の取り扱いについてはすでに述べた。図4に示した生成の過程では，規則は，

語彙 ⟶ 構造 ・・・(7)

の形をしている。

他方，従来の格文法[6]によって文(1)を表現すると，図5(a)のような構造になるが，これからTL(日本語)の構造(図5(d))を得るためには，図5(c)のような規則が必要になる。これは，

構造 ⟶ 構造 ・・・(8)

の形の変換規則であり，効率が悪く，また，規則間の競合も問題にしなければならなくなる。

同様の問題は時制(図6)や部分否定(図7)の場合にも生じる。

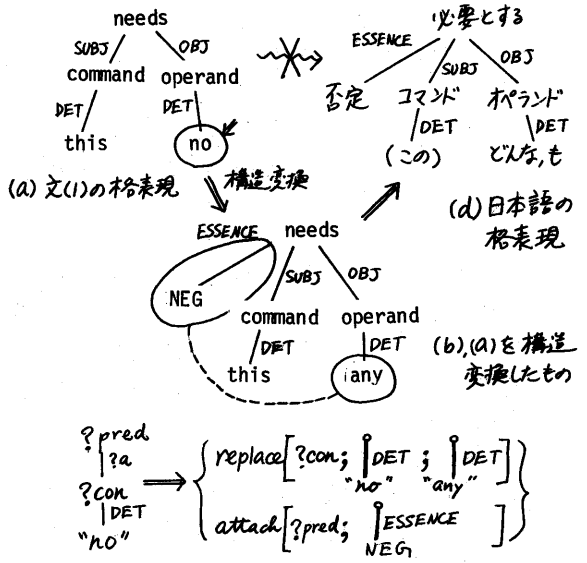


図5. 格表現を中間表現とする翻訳処理 (prenominal negation に対して)

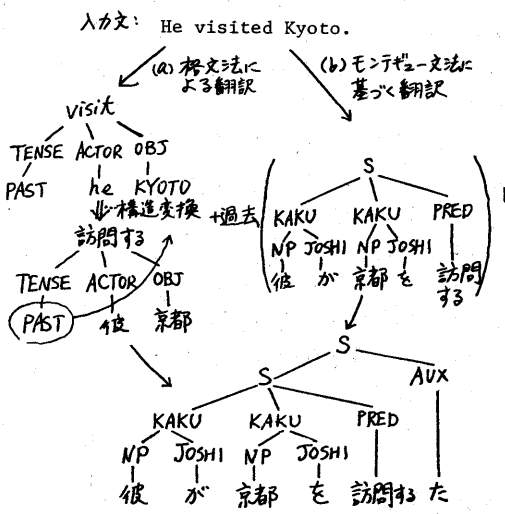


図6. 時制のある文を翻訳する場合

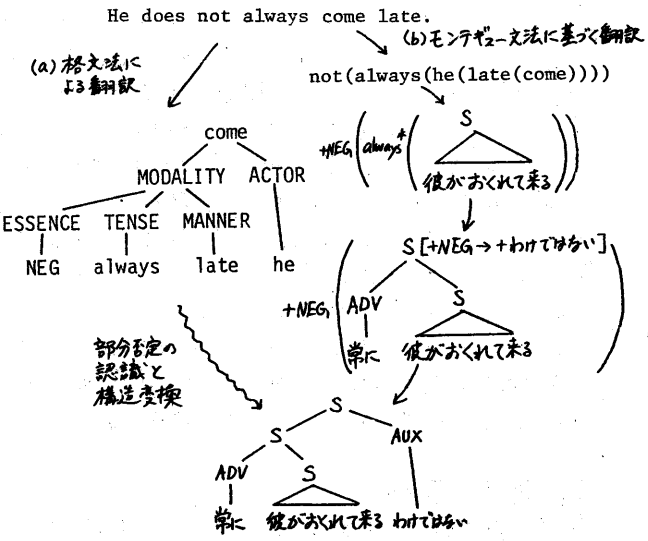


図7. 部分否定を含む文を翻訳する場合

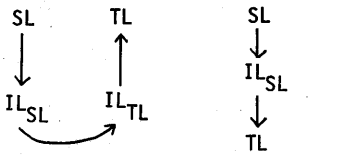
3. システム設計

モンテギュー文法の枠組に基づいた翻訳システムの構成法は2通り考えられる（図8）。

(a) 変換による生成 (generation by translation)

この方式では，各自然言語 L_i に対して，対応する語彙をもつ人工言語（内包論理）IL_i を定義し，L_i を IL_i に写像する翻訳写像とその逆写像を与える。次に，各 IL_i 間で論理式の変換を行なうトランスファー写像を定義する（図8(a)）。

(b) 解釈による生成 (generation by interpretation)



(a) 変換による生成 (b) 解釈による生成

図8. モンテギュー文法の枠組に基づく翻訳システムの構成

この方式では，SL（Source Language）を解析して得られた内包論理式 IL_{SL} の意味解釈に対応する処理を行なって TL の文を得る。具体的には，IL_{SL} の各定数に対して，TL の句構造または句構造生成関数を代入し，それを一つの関数とみなして評価実行することにより TL の文を得る（図8(b)）。

方式(a)は我々がはじめに試作したモデルであり[4]，SL の解析と TL の生成の対称な点が特徴である。また，トランスファーの処理も，高度な翻訳を行なうことを目的としなければ，ほとんど単語の置き換えだけですみ，多国語間翻訳に有効である。一方，方式(b)では，文生成の過程がモンテギュー文法でいう意味解釈に対応しており，より深いレベルでの翻訳に向いていると思われる。

本稿では，方式(b)について述べる。翻訳過程は英文解析の過程と日本語生成の過程に大別される。

英文解析の過程では，拡張文脈自由文法規則（AUGCF規則）という記述法を用いて英文の統語構造と論理式の対応づけを与える。AUGCF規則だけを用いて入力文を解析するパーザは効率が悪いので，各規則の使用法に関する記述を加えることにより得られる手続き的な解析規則によって入力文を解析するようにした。また，このようなシステムは「開いた」性格をもち，直ちに完全な知識が与えられることは期待できない。このような場合，解析規則の制約がゆるすぎて，（人間には）あいまいでないような入力文に対しても多数のあいまい性が出力されてしまうこともおこる。そこで不完全な解析規則によってでも人間が対話的に補助することにより，正しい解析にすばやく到達するための機能や，規則作成のための診断情報が出力できるような解析システムを作成した[5]。

日本語生成の過程で用いる句構造操作関数では従来の変形操作を使用する。変形操作は，例えば＋時制，疑問化など，いわゆる意味を変える変形とそうでないものに分類される。前者は句構造操作関数として組み込む。後者は，スタイルを整えたり，出力文を滑らかにしたりする，一種の「整形」操作であると位置づけられ，生成の各サイクルで働くプロダクションルールとして組み込む。ここでは出力文を一気に生成することはせず，まず，内包論理式から句構造を生成し，次に句構造の終端記号にあたる語彙を順にとり出して，隣接する要素について活用処理を行ないながら，一次元に配列し，出力文を得る。

4. 英文の解析

4.1 拡張文脈自由文法規則による統語と意味の記述

英語の統語と意味の対応づけは，文脈自由文法規則を拡張した形式（AUGCF規

則）と辞書によって行なう。AUGCF規則の形式は，

$$\text{Rn:}\quad A \xrightarrow{f_{syn}, f_{score}, f_{sem}} \beta_1 \cdots \beta_n \qquad \cdots(9)$$

である。ここで，Aは非終端記号であり，文法範疇を表わす。β_iは非終端記号B_iまたは削除項$B_i - C_i$である。$B_i - C_i$は文法範疇B_iの句構造から文法範疇C_iの句構造がちょうど1個削除された構造を表わす。これは関係節における先行詞削除など，変形文法理論でいう削除規則の適用を受けたと考えられる構造を記述するために用いる。

各文法範疇に対応する構造は，文法範疇，syn値，score値，sem値より構成される。f_{syn}, f_{score}, f_{sem}はこれらの値を子の節点から計算する関数である。

図9にサンプル文法を与え，それによる文(1)の解析例を図10に示す。

(a)サンプル文法のAUGCF規則

R1: S $\xrightarrow{\text{subjvp},+10,*\text{sem}_1(*\text{sem}_2)}$ NP.VP

R2: NP $\xrightarrow{\text{pm}_2,+10,*\text{sem}_1(*\text{sem}_2)}$ DET.NOUN

R3: NOUN $\xrightarrow{\text{pm}_2,+10,*\text{sem}_1(*\text{sem}_2)}$ ADJ.NOUN

R4: NP $\xrightarrow{\text{pm}_1,+0,\text{make-np-pp}}$ NP.PP

R5: PP $\xrightarrow{\text{ok},+10,\text{make-prep-np}}$ PREP.NP

R6: NP $\xrightarrow{\text{pm}_1,+0,\text{make-np-rel}}$ NP.WHICH,(S-NP)

R7: VP $\xrightarrow{\text{pm}_1,+10,\lambda x[*\text{sem}_2(\lambda y[*\text{sem}_1(x,y)])]}$ VT.NP

ここでf_{syn}, f_{score}, f_{sem}として用いられた関数の定義は次の通りである：

f_{syn}：ok：統語上の検査は行なわない（無条件に成功させる）
pm_i：i番目の子のsyn値を返す．
subjvp：NPの統語素性とVPの統語素性を検査し，主語と述語動詞の一致を調べ，一致していなければ規則の適用を阻止する．

f_{score}：$+\alpha$：子の節点のscore値をすべて加え，さらにそれにαをかえる．

f_{sem}：$*\text{sem}_i$：i番目の子の節点の意味構造(sem値)をあらわす．
make-np-pp, make-prep-np, make-np-relはそれぞれ適切な意味構造を返すものとする．

(b)サンプル文法の辞書項目の例

```
This:  (DET,(),10,this')
command:  (NOUN,(NBR=SGL,PSN=3),10,command')
needs:  (VT,(TENSE=PRES,FORM=+S),10,need')
no:  (DET,(),10,no')
operand:  (NOUN,(NBR=SGL,PSN=3),10,operand')
```

図9．英語の小さな断片を記述するサンプル文法

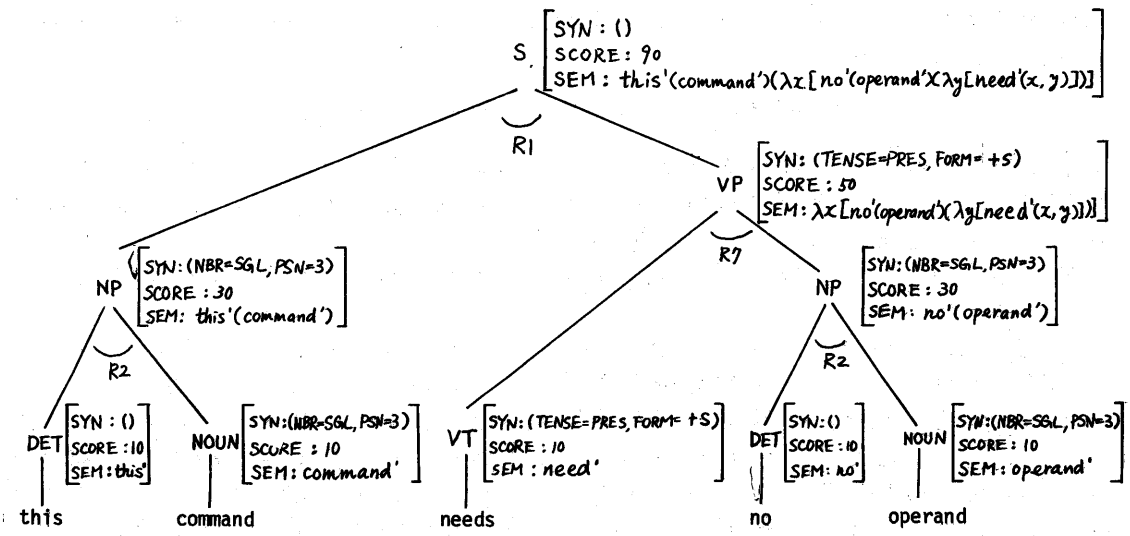


図10．サンプル文法による英文(1)の解析例

4.2 AUGCF規則のATNG型規則への書き換え

AUGCF規則で書かれた文法だけでも比較的小規模なものであれば，それによって直接入力文を解析することもできる。しかしこのような一様なアルゴリズムによる方法では制御に関する知識を組み込めないので文法の規模が大きくなると効率よく働かなくなってしまう。そこで知識工学の手法[2]を参考にし，トップダウン解析を中心とした比較的単純なパージングアルゴリズムを中心とし，各AUGCF規則を手続き的なものに書き換えたものを知識として利用しながらパージングを行なう処理方式を採用した。各AUGCF規則はこの処理過程のどの局面で呼び出されるかに従って，E-rule，U-rule，B-rule，語彙規則の四つのタイプに分類される。

(1) E-rule：生成規則 $A \rightarrow \beta_1 \cdots \beta_m$ においてAがゴールとして予測されたとき呼び出される。記述の形式は次のようである：

$$goal = \langle ゴール \rangle \Rightarrow \cdots ; \langle 条件_i \rangle \rightarrow \langle 手続き_i \rangle ; \cdots . \qquad \cdots (10)$$

E-ruleが呼び出されると，$\langle 条件_i \rangle$として与えられた関数を順に評価し，真になったものについて$\langle 手続き_i \rangle$を起動する。二つ以上の条件が真になれば非決定的処理を行なう。〈条件〉部には任意のLISP関数を与えることができる。

左回帰的でない生成規則をもつAUGCF規則はE-ruleとして記述できる。例えば，サンプル文法のR1は，

```
goal=S ⇒ T → expand[(NP VP);subjvp;+10;*sem1(*sem2)]   ... (11)
```

と書くことができる。(11)はおよそ「Sがゴールとして設定されたとき，（条件部がTなので無条件に）それをサブゴールNPとVPに分解せよ」という意味である。

(2) U-rule：生成規則 $A \rightarrow \beta_1 \cdots \beta_m$ においてβ_iに対応するノードが生成されたとき呼び出される。記述の形式は次のようである：

$$constructed = \langle 文法範疇 \rangle \Rightarrow \cdots ; \langle 条件_i \rangle \rightarrow \langle 手続き_i \rangle ; \cdots . \qquad \cdots (12)$$

左回帰的な生成規則をもつAUGCF規則はU-ruleとして記述できる。例えば，サンプル文法のR4は，

```
constructed=NP ⇒ ?cat[PREP] → consif[(PP);NP;pm1;+0;make-np-pp]; ... .  ... (13)
```

と書き換えられる。(13)はおよそ，「NPがボトムアップされたとき，次の語の文法範疇がPREPであれば，さらにPPを予測し，それが成功すれば，pm_1, +10, make-np-ppを適用して新しいNPノードを構成せよ」という意味である。

関係節（例えばサンプル文法のR6）も同様の処理を行なうが先行詞を保持するという特殊な処理を行なう。

(3) B-rule：任意のAUGCF規則を用いる。単純名詞句のように，入力文の局所的な情報だけからデータ駆動型解析を行なうのが適している場合に用いる。

(4) 語彙規則：特殊な語彙に依存する規則は，辞書中の語の項目の中に分散しておいて，その語が入力文中に出現したとき起動する。例えば，動詞句の解析には，予め種々の動詞句パターンを用意しておいてそれを順次入力文にパターンマッチさせるのではなく，動詞をスキャンしたときに辞書記述から動詞句パターンをとり出し，それによって後続の構造を解析する。このために特殊な手続を用いる。

4.3 対話的診断機能

英文解析プログラムは，与えられた入力文に対して可能な解析を一つずつ順に返してくる。規則に組み込まれた，制約条件としての知識が不完全であると，数多くの解析結果が表示され，なかなか求めるものが得られない。そこで，出力された解析結果を対話的に診断し，誤りを指摘することによってシステムを正しい方向に導けるようにする。図1に入力文に対する診断例の一部を示す。解析プログラムは解析結果が得られる毎に，その要約を出力する。人間はコマンドによって解析結果の木構造の上をスキャンしながら，誤りの部分，正しい部分を指摘していく。ファイルに解析結果を全て出力することも可能である。

```
The asynchronous operation of VERSAbus allows very slow peripherals
to be matched to high speed processors without significant reduction
to the overall system speed.
(cited from: VERSAbus Preliminary Specification, Motorola, 1979)
```

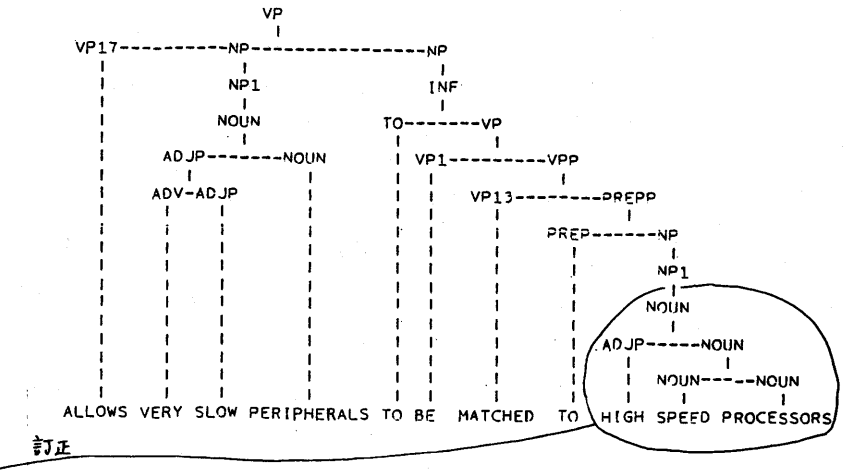
…入力文

⇓

```
> PARSE-1 IS FOUND.        --- 第1番目の解析結果が得られた.
"DIAGNOSTIC MODE"          --- 診断モードであることを示す.
SENTENCE
  DCL
    CLAUSE
      NP
        NP .. THE ASYNCHRONOUS OPERATION
        PREPP .. OF VERSABUS
      VP
        VP .. ALLOWS VERY SLOW PERIPHERALS TO BE MATCHED TO
              HIGH SPEED PROCESSORS
        PREPP .. WITHOUT SIGNIFICANT REDUCTION TO THE
                 OVERALL SYSTEM SPEED
  END .. /
SCAN VP                    --- 「VPをトラバースせよ」
SCAN- VP
VP
  VP .. ALLOWS VERY SLOW PERIPHERALS TO BE MATCHED TO HIGH
        SPEED PROCESSORS
  PREPP .. WITHOUT SIGNIFICANT REDUCTION TO THE OVERALL
           SYSTEM SPEED
"SCAN MODE"
N                          --- 「次(のVP)へ」
SCAN- VP
VP
  VP17 .. ALLOWS
  NP .. VERY SLOW PERIPHERALS
  NP .. TO BE MATCHED TO HIGH SPEED PROCESSORS
"SCAN MODE"
TREE                       --- 「(現在のVPノードの)統語構造を木の形で表示せよ」
```

第1番目の解析結果の要約

PAGE.1




```
"SCAN MODE"
SCAN (NOUN NOUN NOUN)
SCAN- (NOUN NOUN NOUN)
NOUN
  NOUN .. SPEED
  NOUN .. PROCESSORS
"SCAN MODE"
NOGOOD *
OK
```

診断

訂正

```
"SCAN MODE"
SEM                        --- 「(現在のVPノードの)意味構造を表示せよ」
(LAMBDA
  X11
  ((A* (*PL ((VERY SLOW) PERIPHERAL)))
    (LAMBDA
      X12
      ((INF-N
        (LAMBDA
          X9
          ((A* (*PL (HIGH ((*ADJ SPEED) PROCESSOR))))
            (LAMBDA X10 ((*EN3 MATCH-TO) X9 X10)))))
       (LAMBDA X13 (ALLOW-TO X11 X12 X13)))))))
```

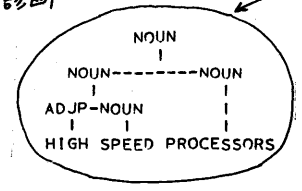


図1. 入力文に対する対話的診断の一例（一部を示す，下線部は人間の入力）

5. 日本語の生成について

図12に句構造および句構造操作関数の抽象化された表現を示す。これにより，内包論理式の各定数に割り当てる句構造または句構造操作関数を記述する。それらはさらに，LISPなどのホスト言語の表現におとされる。

この表記を用いて句構造の生成過程を記述した例を図13に示す。関数の評価実行に必要な規則は，ラムダ計算と変形である。現在用いている変形操作は単純なものがほとんどであるので，変形操作はLISPプログラムとして直接記述されている。

データ構造

句（CPS, conceptual phrase structure と呼ぶ）

表記：【〈カテゴリ〉 〈子3系〉 with … ⟨attr$_i$⟩=⟨value$_i$⟩…】

ただし，〈子3系〉は { CPSの並び：… 【…】 … / 終端記号（語彙項目）：「…」

（例）非終端ノード 【名詞句【限定詞…】【名詞…】with …】

終端ノード 【名詞「犬」with NBR=SGL, CLASS=+ANIM, …】

変数（制限付）

表記：！〈カテゴリ〉（〈変数名〉） （例）！文(s)

句構造操作

単純操作

CPSの合成：𝕀〈カテゴリ〉〈子3系〉with …⟨attr$_i$⟩⇐⟨val$_i$⟩…𝕀

変形操作：f(CPS)，（例）＋否定(【文…】)；文の否定

属性の操作：追加，削除，変更

関数的適用：CPSをactiveな要素として機能させる．例えば，形容詞のように他を修飾すると同時に他から修飾されるものをこのように扱う

条件付操作：【〈条件$_1$〉→〈操作$_1$〉；…】

例：break$_{(動詞)}$ ⇐ λ(x,y) [class(x)=+ANIM→「xがyをこわす」; class(x)=−ANIM→「xによってyがこわれる」]

図12. 句構造および句構造操作関数の表記

```
#dcl((this'(command'))(λx[(no'(operand'))(λy[need'(x,y)])]))  ... (3')

辞書ひき
  #dcl ⇐ #dcl* ≙ λ(x)[【S【文!文(x); 【終助詞「#3」】】, 【E「。」】】]
  this' ⇐ this* ≙ λ(p)[λ(q)[q(p(𝕀名詞句【連体詞「この」】【形式名詞「もの」】𝕀))]]
  command' ⇐ command* ≙ 【名詞「コマンド」】
  no' ⇐ no* ≙ λ(p)[λ(q)[+否定(q(+も(p(【名詞句【連体詞「一つの」】【形式名詞「もの」】】))))]]
  operand' ⇐ operand* ≙ 【名詞「オペランド」】
  need' ⇐ need* ≙ λ(x,y)[【文+格ROLE=SUBJ(!名詞句(x)), +格ROLE=OBJ(!名詞句(y)),
                          【述語【動詞「必要とする」】】】]

#dcl*((this*(command*))(λx[(no*(operand*))(λy[need*(x,y)])]))

実行
  主な実行の規則：
  — λ-計算，主にβ変換，i.e.,（λx[α])(β) → [α]β/x
  — λ(x…)[α] --- 関数値（functional value）としてFUNARG tripleを作る．
  — 【α…】(【β…】) --- 特殊な手続き f$_{αβ}$ を起動し，f$_{αβ}$(【α…】, 【β…】)とする
  — ＋変形(α) --- ＋変形で指定された変形操作の定義を呼出してαに適用する．

適用処理
  【S【文【格【名詞句【連体詞「この」】【名詞「コマンド」】】【助詞「は」】】
      【格【名詞句【連体詞「一つの」】【名詞「オペランド」】】【助詞「も」】】
      【述語【動詞「必要とする」】】】
    【助動詞「ない」】】
  【E 。】
「このコマンドは一つのオペランドも必要としない。」
```

図13. 日本語文生成過程の詳細

6. 実験と検討

以上に述べたシステムを研究室のLISP上に作成した。システムの構成は図14のようになっている。翻訳過程の各段階で必要となる辞書は，LISPの直接アクセスファイル上におかれていて，与えられた入力文に必要なデータのみがコア上にとり込まれる。未知の単語に遭遇すると，直ちにLISPエディタが起動され，その語の辞書項目を登録する。このとき未登録の語に類似の，すでに登録された語の辞書項目を利用して効率的に編集を行なうことができるようになっている。

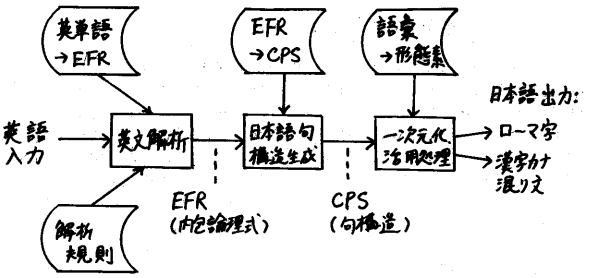


図14. 英文和訳システムの構成（データフロー）
ただし，EFR: English-oriented Formal Representation

実験の第一段階として，基本的な文法現象について翻訳実験を行なった。そのうちの約100例が[3]に示されている。

現在は実験の第二段階として，実際の計算機マニュアルに含まれる文を対象に翻訳実験を行なっている。その例を付録に示す。

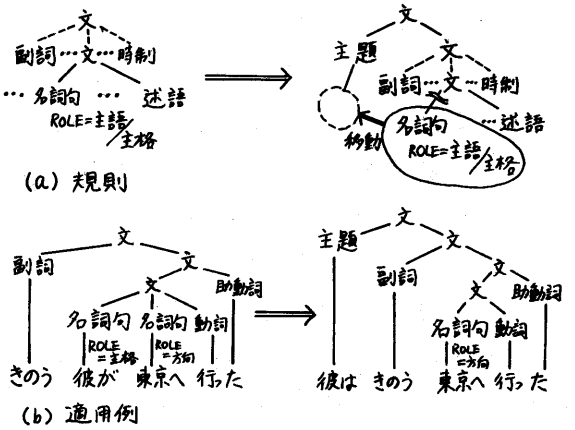


図15. ad hocな規則の例 ── 主語，主格の主題化

現在の目標は翻訳機構の解明を主眼とするものであるので，人間の常識などに基づいてあいまい性を解消するような処理は行なわなかった。そのため，付録に示した各文には数多くのあいまい性（その多くは人間によれば解消される）が生じた。現在はこれを対話処理によって解消しているが，今後の課題である。

現在のシステムに残された第二の課題は日本語生成に関係する。例えば，現在のシステムで生成する日本語文は常に主語が主題化されているが（図15），これはad hocな規則と言える。このようなad hocな規則は，他にも，例えば，aとtheの訳出などに関しても用いられているが，これらの現象をどのように体系化するかが課題として残されている。上のような例に対しては文脈の考慮が必要になってくるであろう。

文献

[1] Dowty: Introduction to Montague Semantics, Reidel, 1981. (井口(他)による翻訳予定)
[2] Feigenbaum: The Art of Artificial Intelligence, Proc. IJCAI-77, 1977.
[3] 清野正樹：英日機械翻訳システムの作成，京都大学修士論文，1981.
[4] 西田，清野，堂下：モンテギュー文法に基づく英文和訳システムの試作，情報処理学会論文誌(to appear).
[5] 西田，堂下：自然言語解析のためのプログラミングシステムCOMPLANについて，情処論文誌(投稿中).
[6] Simmons: Semantic Networks: Their Computation and Use for Understanding English Sentences, in Schank and Colby (eds.) Computer Models of Thought and Language.

(a) 入力の英文テキスト

The assembly language provides a means for writing a program without being concerned with actual memory addresses or machine instruction formats. It allows the use of symbolic addresses to identify memory locations and mnemonic codes to represent the instructions. Labels can be assigned to a particular instruction step in a source program to identify that step as an entry point for use in subsequent instructions. Operands following each instruction represent storage locations, registers, or constant values. The assembly language includes assembler directives that supplement the machine instruction. A pseudo-op is a statement which is not translated into a machine instruction. A pseudo-op is a statement which is interpreted as a directive that controls the assembly process.

A program written in assembly language is called a source program. It consists of symbolic commands called statements. Each statement is written on a single line. The source program is processed by the assembler to obtain a machine language program that can be executed directly by the Z80.

(cited from: Z80-Assembly Language Programming Manual, Zilog, 1977)

(b) 日本語への翻訳結果

アセンブリ言語は実際のメモリアドレスないしは機械命令形式を知ることなしにプログラムを書くための方法を与える．それはメモリロケーションを識別するための記号アドレスとその命令を表現するためのニモニックコードの利用を許す．ラベルはそのステップを後続の命令のなかの利用のためのエントリポイントとして識別するためにソースプログラムのなかの特定の命令ステップに代入されることができる．各命令に続いているオペランドはストレッジロケーションないしはレジスタないしは定数を表現する．アセンブリ言語はその機械命令を補うアセンブラ命令を含む．擬似命令は機械命令に翻訳されないステートメントである．擬似命令はアセンブリ過程を制御する命令と理解されるステートメントである．

アセンブリ言語で書かれたプログラムはソースプログラムと呼ばれる．それはステートメントと呼ばれる記号命令から成る．各ステートメントは一つの行の上に書かれる．そのソースプログラムは直接Ｚ８０によって実行されることができる機械言語プログラムを得るためにアセンブラによって処理される．

付録　試作したシステムによる翻訳例

（ただし，まだシステムの解析力が十分でないため，原文を一部修正している）